Paraconsistent Declarative Semantics for Extended Logic Programs

Ofer Arieli

Report CW 299, September 2000

Katholieke Universiteit Leuven
Department of Computer Science
Celestijnenlaan 200A – B-3001 Heverlee (Belgium)

Paraconsistent Declarative Semantics for Extended Logic Programs

Ofer Arieli

Report CW 299, September 2000

Department of Computer Science, K.U.Leuven

Abstract

We introduce a fixpoint semantics for logic programs with two kinds of negation: an explicit negation and a negation-by-failure. The programs may also be prioritized, that is, their clauses may be arranged in a partial order that reflects preferences among the corresponding rules. This yields a robust framework for representing knowledge in logic programs with a considerable expressive power. The declarative semantics for such programs is particularly suitable for reasoning with uncertainty, in the sense that it pinpoints to the incomplete and inconsistent parts of the data, and regards the remaining information as classically consistent. As such, this semantics allows to draw conclusions in a non-trivial way even in cases that the logic programs under consideration are not consistent. Finally, we show that this formalizm may be regarded as a simple and flexible process for belief revision.

Paraconsistent Declarative Semantics for Extended Logic Programs

Ofer Arieli
Department of Computer Science, K.U.Leuven
Celestijnenlaan 200A, B–3001 Heverlee, Belgium
arieli@cs.kuleuven.ac.be

September, 2000

Abstract

We introduce a fixpoint semantics for logic programs with two kinds of negation: an explicit negation and a negation-by-failure. The programs may also be prioritized, that is, their clauses may be arranged in a partial order that reflects preferences among the corresponding rules. This yields a robust framework for representing knowledge in logic programs with a considerable expressive power. The declarative semantics for such programs is particularly suitable for reasoning with uncertainty, in the sense that it pinpoints to the incomplete and inconsistent parts of the data, and regards the remaining information as classically consistent. As such, this semantics allows to draw conclusions in a non-trivial way even in cases that the logic programs under consideration are not consistent. Finally, we show that this formalizm may be regarded as a simple and flexible process for belief revision.

Keywords: Logic programming, Fixpoint semantics, Paraconsistency, Multy-valued logics.

ACM Classifications:
F.3.2 (Logics and Meanings of Programs/Semantics of Programming Languages),
I.1.3 (Computing Methodologies/Symbolic and Algebraic Manipulation/Languages),
I.2.4 (Artificial Intelligence/Knowledge Representation Formalisms and Methods).

1 Introduction

Our goal in this paper is twofold: To give semantics to logic programs with a strong expressive power, and to provide within this context "proper" means of managing inconsistent and incomplete information. Clearly, both these requirements have direct impacts on the way that knowledge is represented and on the way that it is processed by the underlying formalizm. We therefore begin by considering some aspects of knowledge representation and reasoning in light of our goals.

1.1 Knowledge representation

The fundamental role of knowledge representation in artificial intelligence in general and in logic programming in particular is well acknowledged. No wonder, then, that this is one of the most extensively investigated topics in these areas. Here we manly focus on different ways to represent *negative data* in logic programs, since we believe that this is a key concept for a proper treatment of inconsistency and incompleteness in the context of commonsense reasoning.

It has long been claimed that there are many cases in which only one type of negation is not sufficient for describing a given situation (see, e.g., [9, Section 3.1] and [47] for detailed discussions on this and other related issues). It has therefore been suggested to use two operators for representing two different types of negative information. Several formalisms that give semantics to logic programs with two kinds of negation have been proposed in the literature (see, e.g., [1, 24, 26, 30, 36, 37, 41] and a survey in [9, Section 3]). One kind of negation, denoted here by $\neg$, corresponds to an "explicit" negative data. Its role, like that of negation in classical logic, is to represent counter-information. The other kind of negation, denoted here by `not`, corresponds to a more "implicit" negative data. It is usually used for expressing the fact that the corresponding assertion cannot be proved or decided on the basis of the available information. It is therefore usual to associate this connective with "negation-as-failure" (to prove the corresponding assertion). The different nature of the two kinds of negations is demonstrated in the following example:

Example 1.1 Consider a rule that expresses the fact that "If someone is innocent (s)he cannot be guilty". This rule may be represented by the following implication:

$$\neg\texttt{guilty}(x) \leftarrow \texttt{innocent}(x)$$

I.e., innocence must entail no guilt. On the other hand, a rule such as the following one:

$$\texttt{innocent}(x) \leftarrow \texttt{not guilty}(x)$$

is somewhat less strict. It may be understood as stating that "someone is innocent as long as it has not been proven that (s)he is guilty".

It follows, then, that the two negation operators should be used in different contexts. This is further illustrated in the following example (borrowed from [26]):

Example 1.2 Consider a rule that states that "a school bus may cross a railway tracks if there is no crossing train". This rule may be represented by the following implication:

$$\texttt{cross\_railway\_tracks} \leftarrow \neg\texttt{train\_is\_comming}.$$

However, it should *not* be expressed as follows:

$$\texttt{cross\_railway\_tracks} \leftarrow \texttt{not train\_is\_comming}$$

The reason for this is that the condition in the latter clause holds in cases that there is no information about the presence or the absence of a train. This is a much weaker condition than that of the former clause, which is satisfied only if there is an *explicit* evidence that no train is approaching.

The existence of two different operators that represent different kinds of negative information affects also the way that queries are represented and evaluated. This is so since it is now possible to distinguish between a query that fails because it *does not succeed*, and a query that fails in a stronger sense, that its *negation succeeds.*

As we have already implied, the use of two kinds of negation is also useful for handling inconsistent and incomplete information. It goes without saying that for defining a robust formalism that handles uncertainty, one should at least be able to represent situations in which uncertainty arise. In other words, a plausible framework for managing uncertainty should support the representation the following types of data/knowledge:

- *Inconsistent information.* A representation of contradictory data should be allowed *within the language* (Unlike, e.g., positive logic programs, the syntax of which rules out any possibility of representing contradictions.[1])
- *Partial knowledge.* I.e., the ability to deal *directly* with incomplete information by *explicitly* pointing out to cases in which the data (or the knowledge) is incomplete.
- *(Hierarchy of) exceptions.* I.e., the ability to disregard some piece of information in the presence of another. More generally, making preferences among different rules. Such preferences may be represented either in the programs language itself or in the 'meta-language' (as an additional information, not necessarily represented in a clausal form, and sometimes not even specified by a first-order formulae).

In what follows we shall see how all these different types of knowledge are represented in our framework. Intuitively, it is probably clear already that the strong negation $\neg$ will be useful for representing contradictory data, while the negation-by-failure operator `not` will be useful for representing incomplete data. In addition, we will also allow an additional information, expressed as a "meta-knowledge", which exhibits preferences of certain pieces of information over others.

1.2 Reasoning with uncertainty

Roughly, logic programming is a combination of logic as a representation language and the theory of (constructive) automated deduction. The first aspect was considered with respect to uncertainty in the previous section. Here we address the other aspect within the same context. The two main requirements in this respect are *non-monotonicity* and *paraconsistency* [14]. I.e., partial information should be accompanied with a formalism for default reasoning that can modify its set of conclusions in the light of new data, and the semantics for inconsistent logic programs should not be trivial, that is, inconsistent information should *not* entail every conclusion. We demonstrate these properties in the following examples:

[1] Furthermore, even some formalisms that do allow negations in the clause bodies and heads (e.g., [26, 30, 41]), treat atomic formulae and their negations as two different ways of representing atomic information, so practically a representation of inconsistent information is not possible in this case as well. We shall return to this issue in what follows.

Example 1.3 Consider the following logic program, where p and q are two atomic formulae, and $\mathtt{t}$ is a propositional constant that corresponds to the classical truth value that represents true assertions.

$$\mathcal{P} \;=\; \{q \leftarrow \mathtt{t},\;\; p \leftarrow \mathtt{t},\;\; \neg p \leftarrow \mathtt{not}\ \neg q\}$$

Intuitively, $\mathcal{P}$ may be understood such that both p and q are known to be true, and $\neg p$ is also true provided that it cannot be shown that the negation of q holds. In this interpretation, $\mathcal{P}$ clearly contains an inconsistent information regarding p. However, a paraconsistent formalism should not attach to $\mathcal{P}$ a trivial fixpoint semantics. Moreover, the consistent part of the program ($\{q \leftarrow \mathtt{t}\}$ in the case of $\mathcal{P}$ above) should be easily distinguishable from the inconsistent part (i.e., $\mathcal{P} \setminus \{q \leftarrow \mathtt{t}\}$).

Suppose now that a new datum arrives, and it indicates that if p holds then $\neg q$ must hold as well. The new program is therefore the following:

$$\mathcal{P}' \;=\; \mathcal{P} \;\cup\; \{\neg q \leftarrow p\}$$

Now, the information regarding p becomes consistent (as the condition for concluding $\neg p$ does not hold anymore), while the data regarding q is now inconsistent. A non-monotonic formalism should adapt itself to the new situation. In particular, while the query $\neg p$ should succeed where $\mathcal{P}$ is the underlying program, it should fail w.r.t. $\mathcal{P}'$.

Example 1.4 A robust formalism for reasoning with uncertainty should also be able to handle incomplete information in a plausible way. This is demonstrated in the following example:

$$\mathcal{P} \;=\; \{q \leftarrow \mathtt{t},\;\; p \leftarrow \mathtt{not}\ p\}$$

This time $\mathcal{P}$ contains an incomplete information regarding p. Unlike some formalisms that do not provide any model for this program (e.g., the stable model semantics [25]), we claim that a proper semantics for $\mathcal{P}$ should distinguish between the meaningful data in $\mathcal{P}$ ($\{q \leftarrow \mathtt{t}\}$), and the meaningless data ($\{p \leftarrow \mathtt{not}\ p\}$). Note that the well-founded semantics [46] do provide a plausible solution to this case. In what follows (Section 3.2) we shall use this property of the well-founded semantics for defining our way of handling incomplete data.

As classical logic is neither non-monotonic nor paraconsistent, reasoning with partial or contradictory information should be done in a non-classical way (see, e.g., [5, 6, 10, 11, 31, 33, 38] for some formal methods of doing so). In our case we use *multiple-valued semantics* in which there are particular truth values that correspond to different degrees of contradictions and partial information (see Sections 2.2 and 4.2 below).

The structure of the paper

The rest of this paper is organized as follows: In the next section we present our framework. In particular, we consider some issues that are related to the semantical nature of our formalism, such as showing that Belnap four-valued structure [10, 11] is particularly suitable for representing the kind of information we intend to decode in the logic programs. In Section 3 we introduce our fixpoint theory, first for logic programs without negation-as-failure, and then for the general

case. In Section 4 we further generalize our formalism to cases in which the logic programs under consideration are prioritized, i.e., every clause has its own relative priority over the other clauses. For extending our semantics to the prioritized case we accordingly consider a generalization of the four-valued semantical structure to a boarder family of multiple-valued algebraic structures, called *bilattices* [19, 27, 28]. The semantics that is obtained is then discussed and its main properties are demonstrated by some examples. In section 5 we summarize the main properties of our formalism (also with respect to other related fixpoint semantics), and conclude.

2 Preliminaries

2.1 Logic programs

In what follows p, q, r denote atomic formulae, $l, l_1, l_2, \ldots$ denote literals (i.e., atomic formulae that may be preceded by $\neg$), and $e, e_1, e_2, \ldots$ denote extended literal (i.e. literals that may be preceded by `not`). The complement of a literal l is denoted by $\overline{l}$ (that is, if $l = p$ for some atom p then $\overline{l} = \neg p$, and if $l = \neg p$ then $\overline{l} = p$). As usual in the context of logic programming, we shall deal with formulae in a clause form, as defined below:

Definition 2.1 Let $n, m \geq 0$.

- A *positive* clause is a formula of the form $p \leftarrow p_1, \ldots, p_n$ [2]
- A *standard* clause is a formula of the form $p \leftarrow p_1, \ldots, p_m, \texttt{not}\ p_{m+1}, \ldots, \texttt{not}\ p_{m+n}$
- A *normal* clause is a formula of the form $p \leftarrow l_1, \ldots, l_n$
- A *general* clause is a formula of the form $l \leftarrow l_1, \ldots, l_n$
- An *extended* clause is a formula of the form $l \leftarrow e_1, \ldots, e_n$

Given a clause $l \leftarrow e_1, e_2, \ldots, e_n$, we say that l is the cause *head*, and $e_1, \ldots, e_n$ is the clause *body* (sometimes abbreviated by $Body$). The clause head is also called the *conclusion* (of the clause), and each element in the clause body is called a *condition* (of the clause). The set of the (extended) literals that appear in $Body$ is denoted by $\mathcal{L}(Body)$.

A (possibly infinite) set $\mathcal{P}$ of extended (respectively: positive, standard, normal, general) clauses is called an *extended* (respectively: *positive*, *standard*, *normal*, *general*) *logic program*.

2.2 Some semantical considerations

A. The underlying multy-valued structures

First, we should decide what underlying semantics is most suitable for our intended formalism. It is well-accepted that two-valued semantics is an appropriate semantical framework for positive

[2] Such formulae are also called *definite* clauses or *Horn* clauses.

logic programs. This is so since every positive logic program $\mathcal{P}$ has a unique least Herbrand model, which is identical to the least fixpoint of van-Emden and Kowalski's immediate consequence operator [45] of $\mathcal{P}$. It follows, therefore, that the "intended" semantics of positive logic programs can be captured within the two-valued setting.

Things are getting more complicated when negations may appear in the clause bodies. In such cases a least two-valued model does not always exist (Consider, e.g., $\mathcal{P}=\{p \leftarrow \texttt{not } p\}$), and there are cases in which several minimal two-valued models exist (For instance, $\mathcal{P}=\{p \leftarrow \neg q,\ q \leftarrow \neg p\}$ has two minimal Herbrand models. In one of them p is true and q is false, and in the other one q is true and p is false). One common way to overcome these problems is to consider a minimization w.r.t. a three valued semantics: Fitting's operator [18, 22], based on Kripke/Kleene 3-valued semantics [29] always yields a least fixpoint when applied to normal logic programs, and this is also the case with the three-valued well-founded semantics [46], applied to standard logic programs. Under some further assumption(s) on the syntactical structure of the logic programs under consideration, some other 2-valued and 3-valued fixpoint semantics are uniquely determined. For instance, as shown in [39, 40], every standard logic program that is weakly stratified [39] has a unique weakly perfect model [39], which coincides with its unique stable model [25] and its unique well-founded model [46].

When negations may also appear in the clause heads, the logic programs may be inconsistent, and so unless inconsistency reduces to triviality, neither 2-valued nor 3-valued models can capture the semantics of such programs anymore. Briefly, this is due to the fact that if $\mathcal{P}$ is some general (or extended) logic program, then an "appropriate" semantics for it should be able to distinguish among the following four different cases:

1. p is a positive fact of $\mathcal{P}$ (i.e., the clause $p \leftarrow \texttt{t}$ appears in $\mathcal{P}$),
2. p is a negative fact of $\mathcal{P}$ (i.e., the clause $\neg p \leftarrow \texttt{t}$ appears in $\mathcal{P}$),
3. p is both a positive and a negative fact of $\mathcal{P}$ (i.e., $\{p \leftarrow \texttt{t},\ \neg p \leftarrow \texttt{t}\} \subseteq \mathcal{P}$),
4. Neither p nor $\neg p$ is the head of any clause in $\mathcal{P}$.

Intuitively, while in the first two cases one expects that p would have a classical value (assuming that p does not appear in any other clause head in $\mathcal{P}$), in the latter two cases *two other values* should be attached to p: One for denoting that the data regarding p is inconsistent (as in case 3 above), and the other for denoting that there is an insufficient information regarding p (as in case 4 above).

It follows that in order to capture these four different cases on the semantical level, a semantical structure for general or extended logic programs should contain (at least) four different elements. Probably the best-known structure with this property is Belnap's $\mathcal{FOUR}$ (Figure 1).

Belnap's algebraic structure was introduced in [10, 11] as a semantical tool for representing different states of a reasoner's knowledge (or belief). This structure consists of four truth values: the classical ones (t, f), a truth value $(\bot)$ that intuitively represents lack of information, and a

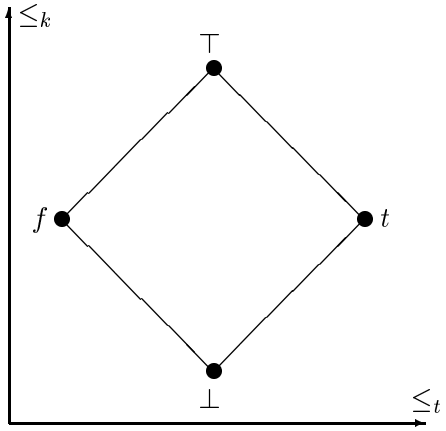

Figure 1: Belnap four-valued structure, $\mathcal{FOUR}$

truth value ($\top$) that may intuitively be understood as representing contradictions. These four elements are simultaneously arranged in two partial orders. In one of them (denoted here by $\leq_t$), f is the minimal element, t is the maximal one, and $\bot, \top$ are two intermediate values that are incomparable. This partial order may be intuitively understood as representing differences in the amount of *truth* of each element. We denote by $\wedge$ and $\vee$ the meet and join operations w.r.t $\leq_t$ (Hence, e.g., $\top \vee \bot = t$). In the other partial order (denoted here by $\leq_k$), $\bot$ is the minimal element, $\top$ is the maximal one, and t, f are two intermediate values. This partial order intuitively represents differences in the amount of *knowledge* (or information) that each element exhibits. We denote by $\otimes$ and $\oplus$ the meet and join operations w.r.t $\leq_k$ (Hence, e.g., $t \oplus f = \top$).

The various semantical notions are defined on $\mathcal{FOUR}$ as natural generalizations of similar classical ones: A *valuation* ν is a function that assigns a truth value in $\mathcal{FOUR}$ to each atomic formula. In what follows we shall sometime write $\nu = \{p : x, q : y\}$ instead of $\nu(p) = x$, $\nu(q) = y$. Any valuation is extended to complex formulae in the obvious way. The set of the four-valued valuations is denoted by V^4.

$\mathcal{D} = \{t, \top\}$ is the set of the *designated* elements of $\mathcal{FOUR}$, i.e., the set of elements in $\mathcal{FOUR}$ that represent true assertions. Hence, we say that a valuation ν *satisfies* a formula ψ iff $\nu(\psi) \in \mathcal{D}$. Note that $\mathcal{D}$ is a prime filter in $\mathcal{FOUR}$ (w.r.t. both $\leq_t$ and $\leq_k$) that consists of the elements that are $\leq_k$-greater than or equal to t. This corresponds to Belnap's observation that the designated elements of $\mathcal{FOUR}$ should be those that are "at least true" (see [11, Page 36]).

A valuation that assigns a designated value to every clause in a logic program $\mathcal{P}$ is a *model* of $\mathcal{P}$.

Next we define a useful partial order on the elements of V^4, using the partial order $\leq_k$ on $\mathcal{FOUR}$:

Definition 2.2

a) A valuation $\nu_1 \in V^4$ is *k-smaller* than another valuation $\nu_2 \in V^4$ if for every atomic formula p, $\nu_1(p) \leq_k \nu_2(p)$.

b) A valuation $\nu \in V^4$ is a *k-minimal* element in a set $S \subseteq V^4$ if there is no other element in S that is k-smaller that ν.

It is easy to see that Definition 2.2(a) induces a lattice structure on the set of the four-valued valuations: $\mathcal{V}^4 = (V^4, \leq_k)$. Another useful way of ordering the elements in V^4 is the following:

Definition 2.3 [5, 6]

a) A valuation $\nu_1 \in V^4$ is *more consistent* than another valuation $\nu_2 \in V^4$ if $\{p \mid \nu_1(p) = \top\} \subseteq \{p \mid \nu_2(p) = \top\}$.

b) A valuation $\nu \in V^4$ is a *most consistent* element in a set $S \subseteq V^4$ if there is no other element in S that is more consistent that ν.

Clearly, the interesting cases of Definitions 2.2(b) and 2.3(b) are obtained when S is the set of the models of the logic program $\mathcal{P}$ under consideration. Two important sets of models are obtained in these cases: The *k-minimal models* of $\mathcal{P}$, and the *most consistent models* of $\mathcal{P}$. We shall reconsider these models in what follows.

B. The meaning of the implication connective

Let $\mathcal{P}$ be a general logic program. The connectives that appear in the bodies or the heads of the clauses in $\mathcal{P}$, i.e.: conjunctions $(,)$[3] and negations $(\neg)$, should be regarded, respectively, as the greatest lower bound and the order-reversing operation w.r.t. the $\leq_t$-partial order of $\mathcal{FOUR}$.[4] This corresponds to the natural extensions for the multiple-valued case of the 2-valued interpretations of these connectives. However, as has already been observed in [9, 26], the implication connective $\leftarrow$ of the program's clauses should *not* be taken as the material implication (denoted henceforth by $\hookleftarrow$). This is so since, for instance, the intuitive meaning of $\{\neg p \leftarrow \mathtt{t},\ p \leftarrow \neg q\}$ is different than the intuitive meaning of $\{\neg p \leftarrow \mathtt{t},\ q \leftarrow \neg p\}$,[5] thus a plausible semantics for $\mathcal{P}$ cannot be 'contrapositive' w.r.t. $\leftarrow$ and $\neg$. Moreover, in the multy-valued setting, the material implication $\hookleftarrow$ is not suitable for representing entailments anymore. This is mainly due to the following reasons:

1. $p \hookleftarrow p$ does not always hold in the four-valued setting, since excluded-middle is not a four-valued tautology (Note that $\nu(p \vee \neg p) = \bot$ when $\nu(p) = \bot$).

2. $\hookleftarrow$ does not have a deductive nature in $\mathcal{FOUR}$. For instance, the fact that every four-valued model of some conjunction *Body* is also a model of a literal l does *not* imply that $l \hookleftarrow Body$ is true in every four-valued valuation (Consider, e.g., the case in which $Body = l$).

We therefore consider an alternative definition for the implication connective, according to which it does function as an entailment in the four-valued setting:

[3] Commas are used here also as a separator among clauses in the same program. This will not cause any ambiguity.

[4] We shall discuss the semantics of the negation as failure operator `not` in a later stage.

[5] Intuitively, in the former program there is no explicit information on the validity of q or $\neg q$, and so q should be assigned $\bot$, while in the latter program the condition in the rule that defines q is satisfied, and so this time q should be assigned t.

Definition 2.4 [5, 7] Let $x, y \in \mathcal{FOUR}$. Define:

$$x \leftarrow y = \begin{cases} x & \text{if } y \in \mathcal{D} \\ t & \text{otherwise} \end{cases}$$

Note that on $\{t, f\}$ the material implication and the new implication are identical, and both of them are generalizations of the classical implication. However, unlike in the material implication, the implication connective defined in 2.4 does preserve both properties of entailments that were mentioned above.

In what follows we therefore use the implication connective of Definition 2.4 for representing the entailments of the program's clauses. Note that the semantics of this implication is in accordance with the following standard way of understanding entailments in logic programs:

Proposition 2.5 For every valuation $\nu \in V^4$, $\nu(l \leftarrow Body) \in \mathcal{D}$ iff either $\nu(l) \in \mathcal{D}$ or $\nu(Body) \notin \mathcal{D}$.

Proof: Immediately follows from Definition 2.4. □

2.3 The language and its extension to the first-order case

The language of the logic programs considered here is based on the implication connective $\leftarrow$, the meaning of which was discussed in the previous section, conjunction that correspond to the $\leq_t$-join operator in $\mathcal{FOUR}$, two negation operators $\neg$ and `not`, and four propositional constants `t`, `f`, `c`, `u`, that are respectively associated with the elements t, f, $\top$, $\bot$ in $\mathcal{FOUR}$. We therefore remain, basically, on the propositional level. However, as first-order clauses are considered as universally quantified, first order logic programs may be handled within our framework as well. We do so by considering their ground instances; Every non-grounded clause is viewed as representing the corresponding set of ground clauses, formed by substituting every variable that appear in this clause with every possible element of the corresponding domain. Formally, let ρ be a ground substitution from the variables of every clause $\mathtt{C}$ in $\mathcal{P}$ to the individuals of the domain of discourse, D, of $\mathcal{P}$. Then we shall consider programs of the form

$$\mathcal{P}^D = \{\rho(\mathtt{C}) \mid \mathtt{C} \in \mathcal{P},\ \rho : var(\mathtt{C}) \to D\}.$$

In what follows we shall abbreviate $\mathcal{P}$ for $\mathcal{P}^D$.

3 Paraconsistent declarative semantics for logic programs

We are now ready to introduce our fixpoint semantics for logic programs. First, we treat general logic programs (I.e., programs without negation-as-failure), and then we consider extended logic programs.

3.1 Semantics for general logic programs

Definition 3.1 Given a general logic program $\mathcal{P}$, define for every $i \geq 1$ and every literal l the following valuations:

$$\nu_0^{\mathcal{P}}(l) = \bot.$$

$$\mathsf{val}_i^{\mathcal{P}}(l) = \begin{cases} t & \text{if } \exists l \leftarrow Body \in \mathcal{P} \text{ s.t. } \nu_{i-1}^{\mathcal{P}}(Body) \in \mathcal{D}, \text{ [6]} \\ \bot & \text{otherwise.} \end{cases}$$

$$\nu_i^{\mathcal{P}}(l) = \mathsf{val}_i^{\mathcal{P}}(l) \oplus \neg\mathsf{val}_i^{\mathcal{P}}(\bar{l}). \text{ [7]}$$

Also, for every propositional constant $\mathtt{x} \in \{\mathtt{t}, \mathtt{f}, \mathtt{c}, \mathtt{u}\}$ that is associated with an element $x \in \{t, f, \top, \bot\}$ in $\mathcal{FOUR}$, we define $\nu_i^{\mathcal{P}}(\mathtt{x}) = \mathsf{val}_i^{\mathcal{P}}(\mathtt{x}) = x \quad (i = 0, 1, \ldots)$.

Note that $\nu_i^{\mathcal{P}}$ behaves as expected w.r.t. negation: Since for every $x, y \in \mathcal{FOUR}$, $\neg(x \oplus y) = \neg x \oplus \neg y$, we have that $\neg\nu_i^{\mathcal{P}}(l) = \neg(\mathsf{val}_i^{\mathcal{P}}(l) \oplus \neg\mathsf{val}_i^{\mathcal{P}}(\bar{l})) = \neg\mathsf{val}_i^{\mathcal{P}}(l) \oplus \mathsf{val}_i^{\mathcal{P}}(\bar{l}) = \nu_i^{\mathcal{P}}(\bar{l})$.

Proposition 3.2 Let $\mathcal{P}$ be a general logic program. Then the sequence $\nu_0^{\mathcal{P}}, \nu_1^{\mathcal{P}}, \nu_2^{\mathcal{P}}, \ldots$, defined in 3.1, is $\leq_k$-monotonic in $\mathcal{V}^4$.

Proof: Since the set $\mathcal{D}$ of the designated values is upwards-closed w.r.t. $\leq_k$, it easily follows from Definition 3.1 that for a given general logic program $\mathcal{P}$, the sequences $\{\mathsf{val}_i^{\mathcal{P}}\}$ and $\{\nu_i^{\mathcal{P}}\}$ are both $\leq_k$-monotonic in i. □

By Knaster-Tarski theorem [44], it follows from Proposition 3.2 that the sequence $\{\nu_i^{\mathcal{P}}\}$ has a $\leq_k$-least fixpoint. Denote this fixpoint by $\nu^{\mathcal{P}}$. An induced consequence relation $|\sim_\nu$ may now be defined as follows: $\mathcal{P} |\sim_\nu \psi$ iff $\nu^{\mathcal{P}}(\psi) \in \mathcal{D}$ (Thus, a query ψ *follows* from a logic program $\mathcal{P}$ if $\nu^{\mathcal{P}}(\psi)$ is designated).

Proposition 3.3 Let $\mathcal{P}$ be a general logic program. Then $\nu^{\mathcal{P}}$ is the k-minimal four-valued model of $\mathcal{P}$. Moreover, it is at least as consistent as any other model of $\mathcal{P}$, and the consequence relation induced by it is paraconsistent.

Proof: See appendix A. □

The last proposition implies, in particular, that $\nu^{\mathcal{P}}$ is a most consistent model of $\mathcal{P}$. As such, it minimizes the amount of inconsistent belief in the set of clauses.[8] This is in accordance with the intuition that while one has to deal with conflicts in a nontrivial way, contradictory data corresponds to inadequate information about the real world, and therefore it should be minimized (see also [6] for a discussion on most consistent models of general theories).

---

[6] $\nu_j^{\mathcal{P}}$ is defined on conjunctive formulae in the usual way: $\nu_j^{\mathcal{P}}(Body) = \bigwedge_{l_i \in \mathcal{L}(Body)} \nu_j^{\mathcal{P}}(l_i)$. Thus, $\nu_j^{\mathcal{P}}(Body) \in \mathcal{D}$ iff $\forall l_i \in \mathcal{L}(Body)\ \nu_j^{\mathcal{P}}(l_i) \in \mathcal{D}$.

[7] Recall that $\bar{l}$ is the complement of l, and $\oplus$ is the $\leq_k$-join operation in $\mathcal{FOUR}$.

[8] Recall Definition 2.3.

Proposition 3.3 also implies that $\nu^{\mathcal{P}}$ minimizes the amount of knowledge that is pre-supposed, i.e. it does not assume anything that is not *really* known. This property is further discussed in Note 3.5 below.

Corollary 3.4 Let $\mathcal{P}$ be a general logic program. Then:

a) $\nu^{\mathcal{P}}$ is the k-least model of $\mathcal{P}$,

b) $\nu^{\mathcal{P}}$ is a most consistent model of $\mathcal{P}$,

c) $\nu^{\mathcal{P}}$ is the k-minimal element among the most consistent models of $\mathcal{P}$.

Proof: Immediately follows from Proposition 3.3 and its proof. □

Note 3.5 Syntactically, normal logic programs are special cases of general logic programs. Still, there is a *semnatical diference* between a set of rules viewed as a normal programs, and the same set of rules viewed as a general (or extended) program. For instance, an absence of an atom p in (most of the) semantics for normal logic programs means that p is false in the model. However, in general or extended logic programs this can be infered from general rules that are stated in the program itself (e.g., by adding rules for closed word assumption, see Example 3.18-a below), and so the absence of p in a general logic program indicates that in the corresponding partial semantics p is *unknown*.[9] For another example on the semantical differences, consider the following (general) program:

$$\mathcal{P} = \{p \leftarrow \neg q,\ q \leftarrow \mathtt{t}\}.$$

Treated as a normal program, some of the 2-valued and the 3-valued fixpoint semantics for $\mathcal{P}$ assign t to q and f to p. According to those semantics the head of a clause program is associated with its corresponding clause bodies, and therefore the truth value attached to a clause head should be the same as the (least upper-bound of) the value(s) of its clause body(ies). This, however, is not the case in our semantics, which assigns t to q and $\bot$ to p. This is justified by the fact that $\mathcal{P}$ is now considered as a *general* logic program, and so had one wanted to identify the information regarding p with that of its clause body, (s)he should have added to $\mathcal{P}$ also the converse implication, i.e., $\neg p \leftarrow q$.[10] In the absence of such clause our four-valued semantics correctly indicates that one should *not* jump to conclusion that p is false! [11]

Once again, this example demonstrates our slogan: Our semantics always assumes *as minimal knowledge as reasonably possible.* Thus, if one wishes to introduce more assumptions (e.g., to apply Clark's completion [13, 32] to specific predicates), (s)he just has to add the appropriate clauses. In the absence of such information the program may have other meaning than what is understood by some of the 2-valued or 3-valued semantics for normal logic programs. Moreover, since it is not always possible to distinguish in standard or normal logic programs among the various possibilities offered by general (or extended) logic programs, such refinements sometimes

[9] See also a remark on this matter in [26, Pages 591–592].

[10] Indeed, as shown in Example 3.10, our 4-valued fixpoint semantics of $\mathcal{P} \cup \{\neg p \leftarrow q\}$ assigns t to q and f to p.

[11] This is so since the condition of the rule that defines p does not hold, and so one cannot infer anything meaningful about p.

cannot even be captured within the 2-valued or the 3-valued semantics for standard/normal logic programs!

In the reminder of this section we show that in certain cases it *is* possible to restore from our 4-valued fixpoint formalism some other 2-valued or 3-valued fixpoint formalisms, if so one wishes.

Proposition 3.6 Let $\mathcal{P}$ be a positive logic program, and let $\mathcal{P}$' be the positive program obtained from $\mathcal{P}$ by replacing every implication connective by a material implication. Denote by $\nu^{\mathcal{P}}_{f/\bot}$ the valuation that is obtained from $\nu^{\mathcal{P}}$ by changing the $\bot$-assignments to f-assignments (I.e., for every atom p, if $\nu^{\mathcal{P}}(p)=\bot$ then $\nu^{\mathcal{P}}_{f/\bot}(p)=f$). Then:

1. $\mathcal{P}$ and $\mathcal{P}'$ have the same classical models (and thus the same least Herbrand model), and
2. $\nu^{\mathcal{P}}_{f/\bot}$ is the (unique) 2-valued minimal Herbrand model of $\mathcal{P}$ and $\mathcal{P}$'.

Proof: See appendix A. □

Note 3.7 The process of restoring Fitting's $\leq_k$-minimal 3-valued Kripke/Kleene semantics for normal logic programs [18] is somewhat more complicated than the process of restoring the 2-valued semantics of positive logic programs, described in Proposition 3.6 above. Note, however, that if $\mathcal{P}$ is a normal logic program, the following properties hold:

1. For every atom p, $\nu^{\mathcal{P}}(p)\in\{t,\bot\}$.

   (*Proof:* For every i and p, $\mathsf{val}_i(p)\in\{t,\bot\}$, and since $\mathcal{P}$ is a normal program, $\mathsf{val}_i(\neg p)=\bot$. Thus, for every i, $\nu^{\mathcal{P}}_i(p)=\mathsf{val}_i(p)\in\{t,\bot\}$, and so $\nu^{\mathcal{P}}(p)\in\{t,\bot\}$ as well).

2. $\nu^{\mathcal{P}}\leq_k\Psi^{\mathcal{P}}$, where $\Psi^{\mathcal{P}}$ is the $\leq_k$-least fixpoint of Fitting's operator for $\mathcal{P}$.

   (*Proof:* By the fact that $\Psi^{\mathcal{P}}$ is a model of $\mathcal{P}$ and $\nu^{\mathcal{P}}$ is the $\leq_k$-least model of $\mathcal{P}$ [12]).

It follows, therefore, that $\nu^{\mathcal{P}}$ can be viewed as an "approximation" of $\Psi^{\mathcal{P}}$: If $\nu^{\mathcal{P}}$ assigns t to some atomic formulae, then so is $\Psi^{\mathcal{P}}$, and if $\nu^{\mathcal{P}}$ assigns $\bot$ to some atom, then $\Psi^{\mathcal{P}}$ assigns either $\bot$ or f to this atom. Thus, in order to restore from $\nu^{\mathcal{P}}$ Fitting's 3-valued fixpoint semantics for $\mathcal{P}$, it is possible to apply Fitting's operator on $\nu^{\mathcal{P}}$ (rather than to start the iterations with a valuation that assigns $\bot$ to every atom), and then to proceed until a fixpoint is reached. This fixpoint coincides with Fitting 3-valued Kripke/Kleene semantics for $\mathcal{P}$.

The next proposition considers some more specific cases in which there is an alternative way of computing Fitting's 3-valued semantics from our 4-valued semantics.

Definition 3.8 Given a normal logic program $\mathcal{P}$, consider the following general logic program:

$$\mathcal{P}^* = \mathcal{P} \cup \{\neg p \leftarrow \overline{l} \mid p \leftarrow Body \in \mathcal{P},\ \nu^{\mathcal{P}}(p)=\bot,\ l\in\mathcal{L}(Body)\} \cup \{\neg p \leftarrow \mathtt{t} \mid p \leftarrow \mathtt{f} \in \mathcal{P},\ \nu^{\mathcal{P}}(p)=\bot\}$$

[12] Note that according to our semantics, the set of models of $\mathcal{P}$ contains the set of models w.r.t. Fitting's semantics. Thus (as illustrated in Note 3.5), although $\Psi^{\mathcal{P}}$ is the $\leq_k$-least model in Fitting's semantics, it is not necessarily the $\leq_k$-least one in our case.

Proposition 3.9 Let $\mathcal{P}$ be a normal logic program in which each atomic formula appears at most once in a clause head. Let also $\Psi^{\mathcal{P}}$ be Fitting fixpoint semantics for $\mathcal{P}$. Then $\Psi^{\mathcal{P}} = \nu^{\mathcal{P}^*}$.

Proof: See appendix A. □

Example 3.10 Consider again the logic program $\mathcal{P}$, given in Note 3.5. By Definition 3.8,

$$\mathcal{P}^* = \{p \leftarrow \neg q, \quad \neg p \leftarrow q, \quad q \leftarrow \mathtt{t}\}$$

By Proposition 3.9, our four-valued semantics for $\mathcal{P}^*$ is the same as that of Fitting 3-valued fixpoint operator for $\mathcal{P}$. In both of them q is assigned t, and p is assigned f.

Note 3.11 The requirement in Proposition 3.9 that every atomic formula should not appear more than once in a clause head is indeed necessary. To see that consider, e.g., the following program:

$$\mathcal{P} = \{q \leftarrow p, \quad q \leftarrow \neg r, \quad r \leftarrow \mathtt{t}\}.$$

Then $\mathcal{P}^* = \mathcal{P} \cup \{\neg q \leftarrow \neg p, \ \neg q \leftarrow r\}$, and while $\Psi^{\mathcal{P}}(q) = \bot$, we have that $\nu^{\mathcal{P}^*}(q) = f$.

3.2 Semantics for extended logic programs

In this section we extend the fixpoint semantics for general logic programs, considered in the previous section, to extended logic programs. So now, in addition to the explicit negation $\neg$, the negation-as-failure operator `not` may also appear in the clauses bodies.

One way of understanding `not` in the four-valued setting is the following: If we don't know anything about p, i.e. we cannot prove either p or $\neg p$, then we cannot say anything about `not` p as well. Otherwise, if p has a designated value in the intended semantics (i.e., p is provable), then `not` p does not hold, and if p does not have a designated value (i.e., it is not provable), then `not` p holds. It follows, then, that `not` $t = f$, `not` $\top = f$, `not` $f = t$, and `not` $\bot = \bot$.[13]

This interpretation of `not` is a natural generalization to the four-valued case of the way `not` is interpreted by the well-founded semantics [46]. We thus give semantics to logic programs in which `not` may appear in the clause bodies by using a 3-valued transformation, similar to that of the well-founded semantics, for reducing extended logic programs to general logic programs. Then we use the machinery of the previous section for giving semantics to the general logic programs that are obtained. Below we formalize this idea.

Definition 3.12 Let ν be a four-valued valuation. The set S_ν of literals that is *associated* with ν is the smallest set that satisfies the following conditions: [14]

$$\text{If } \nu(l) = t \text{ then } l \in S_\nu, \quad \text{if } \nu(l) = f \text{ then } \overline{l} \in S_\nu, \quad \text{if } \nu(l) = \top \text{ then } \{l, \overline{l}\} \subseteq S_\nu.$$

[13] It is interesting to note that in this interpretation, `not` may be represented as a conjunction of two other negation operators: `not` $p = \neg p \wedge \sim p$, where $\sim p$ is an abbreviation of $p \rightarrow f$, i.e., $\sim p = f$ if $p \in \mathcal{D}$, and $\sim p = t$ if $p \notin \mathcal{D}$.

[14] Such sets are sometimes called *answer sets* (for ν). We shall not use this terminology here, since it is usual to require that if an answer set contains a pair of complementary literals, then it should contain *every* literal. Since our formalism does not reduce to triviality in the presence of inconsistent information, this requirement should obviously not hold here.

Obviously, one can define the converse transformation as well: A four-valued valuation ν_S may be constructed from a set S of literals as follows: For every atom p,

$$\nu_S(p) = \begin{cases} t & \text{if } p \in S \text{ and } \neg p \notin S \\ f & \text{if } p \notin S \text{ and } \neg p \in S \\ \top & \text{if } p \in S \text{ and } \neg p \in S \\ \bot & \text{if } p \notin S \text{ and } \neg p \notin S \end{cases}$$

Definition 3.13 Let $\mathcal{P}$ be an extended program and let S be a set of literals. The *reduction* of $\mathcal{P}$ w.r.t. S is the general logic program $\mathcal{P}{\downarrow}S$, obtained from $\mathcal{P}$ as follows:

1. Each clause that has a condition of the form `not` l for some $l \in S$, is deleted from $\mathcal{P}$.
2. Every occurrence of `not` l, where $\overline{l} \in S$, is eliminated from the (bodies of the) remaining clauses.[15]
3. Every occurrence of `not` l in the (bodies of the) remaining clauses is replaced by the propositional constant `u`.[16]

Now we are ready to define our fixpoint semantics for extended logic programs. Recall that $\nu^{\mathcal{P}}$ denotes the fixpoint semantics for a general logic program $\mathcal{P}$.

Definition 3.14 A valuation $\mu \in V^4$ is an *adequate solution* for an extended logic program $\mathcal{P}$, if it coincides with the fixpoint semantics of the general logic program obtained by reducing $\mathcal{P}$ w.r.t. the set that is associated with μ. In other words, μ is an adequate solution for $\mathcal{P}$ iff

$$\mu = \nu^{\mathcal{P}{\downarrow}S_\mu}.$$

Note 3.15 If the only negation operator that appears in $\mathcal{P}$ is $\neg$, then $\mathcal{P}$ is a general logic program, and so its unique adequate solution is $\nu^{\mathcal{P}}$. It follows, in particular, that the notion of adequate solutions of extended logic programs is a generalization of the definition of fixpoint semantics for general logic programs.

Proposition 3.16 An adequate solution for $\mathcal{P}$ is a model of $\mathcal{P}$.

Proof: Let μ be an adequate solution for $\mathcal{P}$, and let $\mathtt{C} = l \leftarrow Body$ be some extended clause in $\mathcal{P}$. We show that $\mu(\mathtt{C}) \in \mathcal{D}$ by an induction on the number of appearances of the operator `not` in $Body$. If `not` does not appear in $Body$, then `C` is a general clause. In this case $\mu(\mathtt{C}) \in \mathcal{D}$ by Proposition 3.3 and Note 3.15. Otherwise, $Body = Body'$, `not` l'. Now,

- If $\mu(l') \in \mathcal{D}$ then $\mu(\mathtt{not}\ l') = f$, thus $\mu(Body) = f$, and so $\mu(\mathtt{C}) \in \mathcal{D}$.
- If $\mu(l') = f$, $\mu(\mathtt{not}\ l') = t$, so $\mu(\mathtt{C}) = \mu(l \leftarrow Body')$. By induction hypothesis $\mu(l \leftarrow Body') \in \mathcal{D}$.

[15] If a clause body becomes empty by this transformation, we treat this body as if it consists of the propositional constant `t`.

[16] Note that for such an l, necessarily $l \notin S$ and $\overline{l} \notin S$.

- If $\mu(l')=\bot$ then $\mu(\texttt{not}\ l')=\bot$. Thus $\mu(Body)=\mu(Body')\wedge\bot\notin\mathcal{D}$, and again $\mu(\texttt{C})\in\mathcal{D}$. □

As it is shown in Example 3.18 below, an extended logic program may have more than one adequate solution, and so one may use different preference criteria for choosing the best solutions among the adequate ones. In the case of general logic programs we have chosen $\leq_k$-minimization as the criterion for prefering the "best" model among the fixpoint valuations. This was justified by the fact that general logic programs may contain contradictory data, and so we want to minimize the redundant information as much as possible. In the present case we rather use the opposite methodology: Since the negation-as-failure operator corresponds to incomplete information, we are dealing here with a lack of data, so this time we should try to restrict the effect of the negation-as-failure operator only to those cases in which indeed there is not enough data available. It follows, therefore, that now we should seek for a *maximal knowledge* (among the adequate solutions).

Definition 3.17 μ is a *most adequate model* of $\mathcal{P}$ if it is a $\leq_k$-maximal adequate solution for $\mathcal{P}$.[17]

Example 3.18

1. $\mathcal{P}=\{\neg p\leftarrow\texttt{not}\ p\}$.
   Intuitively, $\mathcal{P}$ represents a closed word assumption (CWA, [42]) regarding p: In the absence of any evidence for p, assume that $\neg p$ holds. $\mathcal{P}$ has two adequate solutions $\mu_1=\{p:\bot\}$ and $\mu_2=\{p:f\}$. But $\mu_2>_k\mu_1$, and so μ_2 is the most adequate model of $\mathcal{P}$.

2. $\mathcal{P}=\{p\leftarrow\texttt{not}\ p\}$.
   The most adequate model here is $\mu(p)=\bot$. This indeed seems to be the only reasonable interpretation here, and it coincides with the well-founded model [46] of $\mathcal{P}$ (when $\mathcal{P}$ is viewed as a standard logic program). Two-valued semantics, such as Gelfond-Lifschitz stable model semantics [25], do not provide any model for $\mathcal{P}$.

3. [36, Example 2] $\mathcal{P}=\{p\leftarrow\texttt{t},\ \neg p\leftarrow\texttt{not}\ q\}$.
   This program has two adequate solutions: $\mu_1=\{p:t,\ q:\bot\}$ and $\mu_2=\{p:\top,\ q:f\}$. The most adequate model here is μ_2. It reflects our expectation that since q does not follow from $\mathcal{P}$, the knowledge about p is contradictory. Note that according to the semantics given in [36], $\mathcal{P}$ does not have any model, since it contains a contradictory information.

We postpone to a later stage (Section 5) some further discussions on the most adequate models and other formalisms for giving semantics to extended logic programs. First we complete the presentation of our formalism also for the prioritized case.

4 Prioritized logic programs

4.1 Motivation

As Proposition 3.3 and Corollary 3.4 imply, the four-valued fixpoint semantics considered in the previous section has several appealing properties. However, there might be cases in which one

[17] I.e., μ is an adequate solution for $\mathcal{P}$ and there is no other adequate solution for $\mathcal{P}$ that is strictly $\leq_k$-bigger than μ. Note also that by Proposition 3.16, μ is indeed a model of $\mathcal{P}$.

would like to refine the inference mechanism induced by this fixpoint. To see this, consider the following example.

Example 4.1 (Tweety dilemma) Consider the following well-known logic program:

$$\texttt{fly}(x) \leftarrow \texttt{bird}(x) \qquad \neg\texttt{reptile}(x) \leftarrow \texttt{bird}(x) \qquad \texttt{bird}(x) \leftarrow \texttt{penguin}(x)$$
$$\neg\texttt{fly}(x) \leftarrow \texttt{penguin}(x) \qquad \texttt{bird}(\texttt{Tweety}) \leftarrow \texttt{t} \qquad \texttt{penguin}(\texttt{Tweety}) \leftarrow \texttt{t}$$

Denote the above program by $\mathcal{P}$. Then:

$$\nu^{\mathcal{P}}(\texttt{bird}(\texttt{Tweety})) = t, \qquad \nu^{\mathcal{P}}(\texttt{penguin}(\texttt{Tweety})) = t,$$
$$\nu^{\mathcal{P}}(\texttt{reptile}(\texttt{Tweety})) = f, \qquad \nu^{\mathcal{P}}(\texttt{has\_feathers}(\texttt{Tweety})) = \bot,$$
$$\nu^{\mathcal{P}}(\texttt{fly}(\texttt{Tweety})) = \top.$$

While the truth values that are assigned to $\texttt{bird}(\texttt{Tweety})$, $\texttt{penguin}(\texttt{Tweety})$, $\texttt{reptile}(\texttt{Tweety})$ and $\texttt{has\_feathers}(\texttt{Tweety})$ match the intuitive expectations,[18] one usually tends to conclude from $\mathcal{P}$ that Tweety *cannot* fly. However, this conclusion is based on some *further, implicit knowledge, that is not represented in the program*. Such knowledge is, e.g., the fact that the rule "birds can fly" has exceptions, which should "override" the default rule. Another kind of knowledge that is not encoded in this program is the fact that the information that Tweety is a penguin is more specific than the statement that it is a bird, therefore the former data should have a higher priority than the latter one, in case of "collisions" between the two.

The above inaccurate conclusion about the flying ability of Tweety is therefore an outcome of the limited way knowledge is represented here rather than a consequence of a faulty reasoning process. A general method to improve knowledge representation is to provide a way to prefer a certain data over the other. In example 4.1, for instance, such mechanism will allow us to indicate that the clause that states that "penguins cannot fly" should get a precedence over the one that states that "birds can fly".

Several methodologies for making such preferences have been proposed in the literature. In [30], for instance, rules with negative conclusions are viewed as representing exceptions of rules with the positive counterparts as their conclusions. As such, the former rules are given higher priorities over the latter rules. Thus, for instance, in the semantics of [30] for $\mathcal{P} = \{\neg p \leftarrow \texttt{t},\ p \leftarrow \texttt{not}\ q\}$, p is *false* (Cf. Example 3.18, Item 3).

In the formalism, proposed by Pereira et al. in [37], preferences of different rules are encoded within the language itself. According to this approach the conflict regarding the flying ability of Tweety in Example 4.1 is resolved by stating that birds can fly unless they "abnormal birds". I.e., the rule $\texttt{fly}(x) \leftarrow \texttt{bird}(x)$ in the program of Example 4.1 is replaced by the following two rules:

$$\texttt{fly}(x) \leftarrow \texttt{bird}(x), \texttt{not abnormal\_bird}(x) \qquad \texttt{abnormal\_bird}(x) \leftarrow \texttt{bird}(x), \neg\texttt{fly}(x)$$

[18] The assignment of $\bot$ to $\texttt{has\_feathers}(\texttt{Tweety})$ is justified by the fact that nothing is mentioned in the program about the property "has_feathers".

In the same paper, Pereira et al. also propose to associate a different 'label' to each program rule, and to insert this label as another condition to the body of the rule. This enables an easy way to represent a hierarchy of rules in the language itself. For instance, the fact that under the conditions specified in $Body$ one should apply a rule labelled by l_1 instead of a rule labelled by l_2, is encoded by the following special preference rule: $\neg l_2 \leftarrow Body, l_1$.

The formalisms mentioned above, although being elegant ones, have their own limitations. First, they rule out any representation of contradictions in the reasoner's belief. Such contradictions do occur in practical problems, and it may be useful to use a methodology to trace them and to represent their effect on the obtained semantics. Second, as already observed in [37], because of the inherent asymmetry in the representation of the hierarchy of exceptions, each time that exception to an exception is made, previous rules in the program should be changed. Third, the rule labelling and the need to maintain the preferences and the exceptions with special additional rules, require a lot of overhead in the level of knowledge representation; In practical cases this might yield awkward programs, in which it would be difficult to grasp the essence from the whole data.

Here we consider another way of making preferences among programs clauses, which has a more quantitative nature. The idea is to attach, in a meta-language, different priorities to different clauses. We do so by assigning to every clause a 'confidence factor' that reflects its relative priority over the other clauses. For this, we consider algebraic structures that generalize Belnap's four-valued structure. In particular, we extend the four-valued semantics to a more general semantics that is based on arbitrarily many truth values. In the next section we review the basic notions that are related to these structures, and in Section 4.3 we use them for giving semantics to prioritized logic programs.

4.2 Bilattices and logical bilattices – An overview

Definition 4.2 [27, 28] A *bilattice* is a structure $\mathcal{B} = (B, \leq_t, \leq_k, \neg)$ such that B is a nonempty set containing at least two elements, $(B, \leq_t)$ and $(B, \leq_k)$ are complete lattices, and $\neg$ is a unary operation on B that has the following properties: (i) if $a \leq_t b$ then $\neg a \geq_t \neg b$, (ii) if $a \leq_k b$ then $\neg a \leq_k \neg b$, (iii) $\neg\neg a = a$.

The original motivation of Ginsberg for using bilattices was to provide a uniform approach for a diversity of applications in artificial intelligence. In particular, he treated first order theories and their consequences, truth maintenance systems, and formalisms for default reasoning. The algebraic structure of bilattices has been further investigated by Fitting and Avron [8, 20, 23]. In a series of paper Fitting has also shown that bilattices are very useful tools for providing semantics for logic programs: He proposed an extension of Smullyan's tableaux-style proof method to bilattice-valued programs, and showed that this method is sound and complete with respect to a natural generalization of van-Emden and Kowalski's operator [19, 21]. Fitting also introduced a multy-valued fixpoint operator for providing bilattice-based stable models and well-founded semantics for logic programs [22]. A well-founded semantics for logic programs that is based on a

specific bilattice (denote here by *NINE*, see Figure 2 below) is considered also in [17]. Bilattices have also been found useful for model-based diagnostics [28], computational linguistics [35], reasoning with inconsistent knowledge-bases [5, 43], and processing of distributed knowledge [34].

As in the four-valued case, we shall continue to denote by $\wedge, \vee, \neg$ the meet, join, and negation operations w.r.t. $\leq_t$, and by $\otimes, \oplus$ the meet and the join operations w.r.t. $\leq_k$. The $\leq_t$-maximal (respectively, $\leq_t$-minimal) element is denoted by t (respectively, f), and the $\leq_k$-maximal (respectively, $\leq_k$-minimal) element is denoted by $\top$ (respectively, $\bot$).

Definition 4.3 [5] Let $\mathcal{B}=(B, \leq_t, \leq_k, \neg)$ be a bilattice.

a) A *bifilter* of $\mathcal{B}$ is a nonempty proper subset $\mathcal{D} \subset B$, such that:
(i) $a \wedge b \in \mathcal{D}$ iff $a \in \mathcal{D}$ and $b \in \mathcal{D}$, (ii) $a \otimes b \in \mathcal{D}$ iff $a \in \mathcal{D}$ and $b \in \mathcal{D}$.

b) A bifilter $\mathcal{D}$ is called *prime*, if it also satisfies the following conditions:
(i) $a \vee b \in \mathcal{D}$ iff $a \in \mathcal{D}$ or $b \in \mathcal{D}$, (ii) $a \oplus b \in \mathcal{D}$ iff $a \in \mathcal{D}$ or $b \in \mathcal{D}$.

Clearly, for every prime bifilter $\mathcal{D}$ we have that $t, \top \in \mathcal{D}$, while $f, \bot \notin \mathcal{D}$.

Definition 4.4 [5] A *logical bilattice* is a pair $(\mathcal{B}, \mathcal{D})$, in which $\mathcal{B}$ is a bilattice and $\mathcal{D}$ is a prime bifilter of $\mathcal{B}$.

The basic semantical notions of the four-valued case can easily be extended to the bilattice-valued case. For instance, given a logical bilattice $(\mathcal{B}, \mathcal{D})$, the notions of valuations, models, etc. are the same as in the four-valued case. The definitions of the implication connective $\leftarrow$ also remains the same: For every $x, y, \in B$ the value of $x \leftarrow y$ is x if $y \in \mathcal{D}$, and it is t otherwise. The only difference is that instead of taking $\mathcal{D} = \{t, \top\}$ as the set of the designated values, we now allow that any prime bifilter in B would be the set of the designated values.

The minimal logical bilattice is $\mathcal{FOUR}$ with $\mathcal{D} = \{t, \top\}$. Next we describe a general way of constructing logical bilattices with arbitrarily many elements:

Definition 4.5 [28] Let $(L, \leq_L)$ be a complete lattice. The structure $L \odot L = (L \times L, \leq_t, \leq_k, \neg)$ is defined as follows:
$(y_1, y_2) \geq_t (x_1, x_2)$ iff $y_1 \geq_L x_1$ and $y_2 \leq_L x_2$,
$(y_1, y_2) \geq_k (x_1, x_2)$ iff $y_1 \geq_L x_1$ and $y_2 \geq_L x_2$,
$\neg(x_1, x_2) = (x_2, x_1)$.

A pair $(x, y) \in L \odot L$ may intuitively be understood so that x represents the amount of belief *for* some assertion, and y is the amount of belief *against* it.

Notation 4.6 Let $(x, y) \in L \odot L$. Denote: $[(x, y)]_T = x$ and $[(x, y)]_F = y$.

Example 4.7 Let $\mathcal{TWO} = (\{0,1\}, 0<1\}$ be the classical (two-valued) lattice. Then in the notations of Definition 4.5, Belnap four-valued bilattice $\mathcal{FOUR}$ is isomorphic to $\mathcal{TWO} \odot \mathcal{TWO}$ by the following isomorphism: t corresponds to $(1,0)$, f corresponds to $(0,1)$, $\bot$ corresponds to $(0,0)$, and $\top$ corresponds to $(1,1)$.

Proposition 4.8 [19, 28] For every complete lattice $(L, \leq_L)$, the structure $L \odot L$ is a bilattice.

Outline of proof: Given a lattice L with a meet operation $\sqcap$ and a join operation $\sqcup$, the bilattice operations are defined as follows:

$$(x_1, y_1) \vee (x_2, y_2) = (x_1 \sqcup x_2, y_1 \sqcap y_2), \qquad (x_1, y_1) \wedge (x_2, y_2) = (x_1 \sqcap x_2, y_1 \sqcup y_2),$$
$$(x_1, y_1) \oplus (x_2, y_2) = (x_1 \sqcup x_2, y_1 \sqcup y_2), \qquad (x_1, y_1) \otimes (x_2, y_2) = (x_1 \sqcap x_2, y_1 \sqcap y_2),$$
$$\neg(x, y) = (y, x).$$

It is easy to verify that for every two elements $x, y, \in L \times L$, $x \vee y$ (respectively, $x \wedge y$) is the least upper bound (respectively, the greatest lower bound) of x and y w.r.t. the $\leq_t$-partial order (Definition 4.5). Similarly, $x \oplus y$ (respectively, $x \otimes y$) is the least upper bound (respectively, the greatest lower bound) of x and y w.r.t. the $\leq_k$-partial order (Definition 4.5), and $\neg$ satisfies all the requirements from a negation operator (Definition 4.2). □

Proposition 4.9 [5] Let $(L, \leq_L)$ be a complete lattice with a maximal element, m. Then the smallest (prime) bifilter in $L \odot L$ is of the form $\{(m, x) \mid x \in L\}$. We shall denote this bifilter by $\mathcal{D}_{L \odot L}$.

Corollary 4.10 Let $(L, \leq_L)$ be a complete bounded lattice. Then

a) $(L \odot L, \mathcal{D}_{L \odot L})$ is a logical bilattice.

b) Every complete bounded lattice can be turned into a logical bilattice.

Proof: Part (a) follows from Propositions 4.8 and 4.9. Part (b) follows from part (a). □

4.3 Bilattice-based semantics for prioritized logic programs

For giving semantics to prioritized logic programs, we have found it useful to concentrate on bilattices of the form $\{0, 1, \ldots, m\} \odot \{0, 1, \ldots, m\}$. We shall denote these bilattices by $\mathcal{B}^m$.

As in the non-prioritized case, we start by considering a semantics to prioritized logic programs without negation-as-failure, and then consider the general case.

Definition 4.11 An *m-prioritized general logic program* is a set of quantitative general clauses, i.e., a set of formulae of the form $l \overset{n}{\leftarrow} Body$, where l is a literal, $Body$ is a conjunction of literals, and n is a number between 1 and m.

The quantitative values of the clauses (the n's) may be intuitively understood as representing "confidence factors" or "threshold values" of (the belief in) the corresponding clauses. The idea is that a head of a quantitative clause is evaluated only if there is a "sufficient" evidence in favour of the clause's body, and the evidence for the complement of the clause head does not exceed the clause's threshold value. Next we formalize this intuition:

Definition 4.12 Given a logical bilattice $(\mathcal{B}^m, \mathcal{D})$ and an m-prioritized general logic program $\mathcal{P}$, consider for every literal l and $i \geq 1$ the following functions:

$\nu_0^{\mathcal{P}}(l) = \mathsf{val}_0^{\mathcal{P}}(l) = \bot$.

$\mathsf{threshold}_i^{\mathcal{P}}(l) = lub_{\leq_k}\{\nu_{i-1}^{\mathcal{P}}(Body) \mid l \overset{n}{\leftarrow} Body \in \mathcal{P}\}$. [19]

$\mathsf{belief}_i^{\mathcal{P}}(l) = lub_{\leq_k}\{\nu_{i-1}^{\mathcal{P}}(Body) \mid l \overset{n}{\leftarrow} Body \in \mathcal{P},\ [\mathsf{threshold}_i^{\mathcal{P}}(\bar{l})]_T \leq n\}$. [20]

$\mathsf{val}_i^{\mathcal{P}}(l) = lub_{\leq_k}(\mathsf{val}_{i-1}^{\mathcal{P}}(l), \mathsf{belief}_i^{\mathcal{P}}(l))$.

$\nu_i^{\mathcal{P}}(l) = ([\mathsf{val}_i^{\mathcal{P}}(l)]_T, [\mathsf{val}_i^{\mathcal{P}}(\bar{l})]_T)$.

Again, for every propositional constant $\mathtt{x}$ that corresponds to an element $x \in \mathcal{B}^m$ we define, for every $i \geq 0$, $\nu_i^{\mathcal{P}}(\mathtt{x}) = \mathsf{val}_i^{\mathcal{P}}(\mathtt{x}) = x$.

In each iteration we therefore compute, for each literal, its 'threshold value', which is the least upper bound (w.r.t. $\leq_k$) of the values attached to the relevant clause bodies at the previous iteration.[21] Then we compute the amount the 'belief' in a certain literal l during the current iteration. Again, the required value obtains by considering the relevant clause bodies, but this time we take into account only those clauses with a 'sufficiently high' confidence factor, i.e., those clauses with l as their head, and with a confidence value that is not smaller than the threshold value of l's complement. $\mathsf{val}_i^{\mathcal{P}}(\cdot)$ is a $\leq_k$-monotonic function that is based on these belief values, and finally — as in the four-valued case — we use $\mathsf{val}_i^{\mathcal{P}}(\cdot)$ for constructing the $\leq_k$-monotonic sequence of valuations $\nu_i^{\mathcal{P}}$ that yields, eventually, the fixpoint semantics for $\mathcal{P}$.

Note 4.13 As in the four-valued case, we have that for every i and l, $\nu_i^{\mathcal{P}}(\bar{l}) = \neg\nu_i^{\mathcal{P}}(l)$.

As the next proposition shows, the semantics for non-prioritized programs (Definition 3.1) is a particular case of the semantics for prioritized programs (Definition 4.12).

Proposition 4.14 Let $\mathcal{P}$ be a general logic program, and let $\mathcal{P}'$ be the 1-prioritized logic program obtained from $\mathcal{P}$ by assigning the quantitative factor $n=1$ to every general clause in $\mathcal{P}$. Then, for every i, $\nu_i^{\mathcal{P}}$ (defined in 3.1) is the same as $\nu_i^{\mathcal{P}'}$ (defined in 4.12).

Proof: See appendix A. □

[19] If there is no clause of the form $l \overset{n}{\leftarrow} Body$ in $\mathcal{P}$, define $\mathsf{threshold}_i^{\mathcal{P}}(l) = \bot$.

[20] If no clause of the form $l \overset{n}{\leftarrow} Body$ is in $\mathcal{P}$, or $[\mathsf{threshold}_i^{\mathcal{P}}(\bar{l})]_T > \max\{n \mid l \overset{n}{\leftarrow} Body \in \mathcal{P}\}$, define $\mathsf{belief}_i^{\mathcal{P}}(l) = \bot$.

[21] This value may be intuitively understood as an 'a-priory' belief in the literal under consideration, since the confidence factors of the clauses are not taken into account in the calculation of this value.

As in the four-valued case, the partial order $\leq_k$ on $\mathcal{B}$ can be used for defining a partial order on the set $V^{\mathcal{B}}$ of the B-valued valuations: A valuation $\nu_1 \in V^{\mathcal{B}}$ is *k-smaller* than another valuation $\nu_2 \in V^{\mathcal{B}}$ (notation: $\nu_1 <_k \nu_2$) if $\nu_1(p) \leq_k \nu_2(p)$ for every atom p. The pair $\mathcal{V}^{\mathcal{B}} = (V^{\mathcal{B}}, \leq_k)$ is clearly a lattice. Moreover,

Proposition 4.15 Let $\mathcal{P}$ be an m-prioritized general logic program, and let $\mathcal{B} = \{0, 1, \ldots, m\} \odot \{0, 1 \ldots, m\}$. The sequence $\nu_0^{\mathcal{P}}, \nu_1^{\mathcal{P}}, \nu_2^{\mathcal{P}}, \ldots$ is $\leq_k$-monotonic in $\mathcal{V}^{\mathcal{B}}$.

Proof: By definition 4.12, for every $i \geq 1$ and every literal l, $\mathsf{val}_{i+1}(l) \geq_k \mathsf{val}_i(l)$. Thus, for every $i \geq 0$ and every l, $\nu_{i+1}^{\mathcal{P}}(l) \geq_k \nu_i^{\mathcal{P}}(l)$, and so $\nu_{i+1}^{\mathcal{P}} \geq_k \nu_i^{\mathcal{P}}$. □

Again, by Knaster-Tarski theorem [44], it follows that $\{\nu_i^{\mathcal{P}}\}$ has a $\leq_k$-least fixpoint. We denote it by $\nu^{\mathcal{P}}$.

The following result is an immediate consequence of Proposition 4.14.

Proposition 4.16 Let $\mathcal{P}$ be a general logic program, and let $\mathcal{P}'$ be the 1-prioritized program obtained from $\mathcal{P}$ by setting $n = 1$ as the quantitative factor of every general clause in $\mathcal{P}$. Then $\nu^{\mathcal{P}}$ (the $\leq_k$-least fixpoint of the $\nu_i^{\mathcal{P}}$-sequence, defined in 3.1) is the same as $\nu^{\mathcal{P}'}$ (the $\leq_k$-least fixpoint of the $\nu_i^{\mathcal{P}'}$-sequence, defined in 4.12).

We now generalize our formalism to m-prioritized *extended* logic programs. We do so in a way which is completely analogous to the way we generalized the fixpoint semantics for non-prioritized general logic program to non-prioritized extended logic programs. I.e., we use a transformation like that of the well-founded semantics to eliminate the negation-as-failure operators from the clauses bodies. What remains is an m-prioritized general logic programs, to which we give semantics in the way described above. The following definitions formalize this process.

Definition 4.17 An *m-prioritized extended logic program* is a set of quantitative extended clauses, i.e.: a set of formulae of the form $l \overset{n}{\leftarrow} Body$, where l is a literal, $Body$ is a conjunction of extended literals, and n is a number between 1 and m.

Definition 4.18 Let $\mathcal{P}$ be an m-prioritized extended logic program.

a) A valuation $\mu \in V^4$ is an *adequate solution* for $\mathcal{P}$, if it coincides with the fixpoint semantics of the m-prioritized general logic program obtained by reducing $\mathcal{P}$ w.r.t. the set that is associated with μ. I.e., $\mu = \nu^{\mathcal{P}\downarrow S_\mu}$.

b) A *most adequate model* of $\mathcal{P}$ is a $\leq_k$-maximal adequate solution for $\mathcal{P}$.

By Proposition 4.14 and Definition 4.18 we have the following result:

Proposition 4.19 Let $\mathcal{P}$ be an extended logic program, and let $\mathcal{P}'$ be the 1-prioritized extended logic program obtained from $\mathcal{P}$ by assigning the quantitative factor $n=1$ to every extended clause in $\mathcal{P}$. Then ν is an adequate solution for $\mathcal{P}$ (according to Definition 3.14) iff it is an adequate solution for $\mathcal{P}'$ (according to Definition 4.18).

4.4 Tweety dilemma, revisited

Consider again the logic program for Tweety dilemma, given in Example 4.1. The corresponding 1-prioritized program is the following:

$$\mathcal{P}_1 = \left\{ \begin{array}{lll} \texttt{fly}(x) \overset{1}{\leftarrow} \texttt{bird}(x) & \neg\texttt{reptile}(x) \overset{1}{\leftarrow} \texttt{bird}(x) & \texttt{bird}(x) \overset{1}{\leftarrow} \texttt{penguin}(x) \\ \neg\texttt{fly}(x) \overset{1}{\leftarrow} \texttt{penguin}(x) & \texttt{bird}(\texttt{Tweety}) \overset{1}{\leftarrow} \texttt{t} & \texttt{penguin}(\texttt{Tweety}) \overset{1}{\leftarrow} \texttt{t} \end{array} \right.$$

Since $\mathcal{P}_1$ is a "flat" program (each clause is assigned the same priority), then by Proposition 4.16 its fixpoint is the same as in the non-prioritized case. In particular, still $\nu^{\mathcal{P}_1}(\texttt{fly}(\texttt{Tweety})) = \top$. However, now it is possible to give different precedence to different clauses. As we have noted during the previous discussion on this example, the clause $\texttt{bird}(x) \leftarrow \texttt{penguin}(x)$ describes only a default property of birds, while the other clauses of the program describe rules that do not have exceptions. We therefore attach to $\texttt{bird}(x) \leftarrow \texttt{penguin}(x)$ a lower priority (confidence factor) than the other assertions. The logic program that is obtained is the following:

$$\mathcal{P}_2 = \left\{ \begin{array}{lll} \texttt{fly}(x) \overset{1}{\leftarrow} \texttt{bird}(x) & \neg\texttt{reptile}(x) \overset{2}{\leftarrow} \texttt{bird}(x) & \texttt{bird}(x) \overset{2}{\leftarrow} \texttt{penguin}(x) \\ \neg\texttt{fly}(x) \overset{2}{\leftarrow} \texttt{penguin}(x) & \texttt{bird}(\texttt{Tweety}) \overset{2}{\leftarrow} \texttt{t} & \texttt{penguin}(\texttt{Tweety}) \overset{2}{\leftarrow} \texttt{t} \end{array} \right.$$

The corresponding bilattice, $\mathit{NINE} = \{0,1,2\} \odot \{0,1,2\}$ is displayed in Figure 2 (see also [5, 6, 43]). We abbreviate its elements with the following notations:

$$\begin{array}{lllll} \bot = (0,0) & \mathit{df} = (0,1) & \mathit{dt} = (1,0) & \mathit{f} = (0,2) & \mathit{t} = (2,0) \\ \mathit{d}\top = (1,1) & \mathit{of} = (1,2) & \mathit{ot} = (2,1) & \top = (2,2) & \end{array}$$

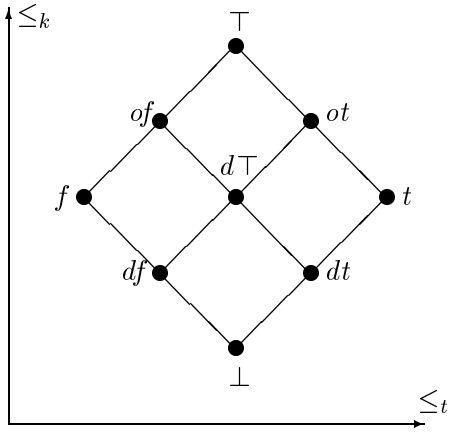

Figure 2: *NINE*

Note that *NINE* has *two* prime bifilters: $\mathcal{D}_t = \{x \mid x \geq_k t\}$ and $\mathcal{D}_{dt} = \{x \mid x \geq_k dt\}$. Consequently, two corresponding logical bilattices may be considered: $\mathcal{NIN}\mathcal{E}_t = (\mathit{NINE}, \mathcal{D}_t)$ and $\mathcal{NIN}\mathcal{E}_{dt} = (\mathit{NINE}, \mathcal{D}_{dt})$. As $\mathcal{D}_t \subset \mathcal{D}_{dt}$, the former logical bilattice may be used for a more skeptical

reasoning process, while the latter one provides a more liberal approach for a query evaluation (we shall demonstrate this in what follows).

Table 1 describes the iterative construction of $\nu^{\mathcal{P}_2}$.

Table 1: iterative construction of $\nu^{\mathcal{P}_2}$

function	`bird`	$\neg$`bird`	`penguin`	$\neg$`penguin`	`fly`	$\neg$`fly`	`reptile`	$\neg$`reptile`
$\mathsf{threshold}_1$	t	$\bot$	t	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$
belief_1	t	$\bot$	t	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$
val_1	t	$\bot$	t	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$
ν_1	t	f	t	f	$\bot$	$\bot$	$\bot$	$\bot$
$\mathsf{threshold}_2$	t	$\bot$	t	$\bot$	t	t	$\bot$	t
belief_2	t	$\bot$	t	$\bot$	$\bot$	t	$\bot$	t
val_2	t	$\bot$	t	$\bot$	$\bot$	t	$\bot$	t
ν_2	t	f	t	f	f	t	f	t

It is easy to see that for every $i \geq 2$, $\nu^{\mathcal{P}_2}_{i+1} = \nu^{\mathcal{P}_2}_i$, thus the $\leq_k$-least fixpoint of $\mathcal{P}_2$ is the following:

$$\nu^{\mathcal{P}_2}(\texttt{bird(Tweety)}) = t, \qquad \nu^{\mathcal{P}_2}(\texttt{penguin(Tweety)}) = t,$$
$$\nu^{\mathcal{P}_2}(\texttt{reptile(Tweety)}) = f, \qquad \nu^{\mathcal{P}_2}(\texttt{has\_feathers(Tweety)}) = \bot,$$
$$\nu^{\mathcal{P}_2}(\texttt{fly(Tweety)}) = f.$$

So the intuitive conclusion regarding the flying ability of Tweety is obtained, and the other literal conclusions remain as in the non-prioritized case, as expected.

It is interesting to note that our approach supports a very flexible process of belief revision. To see this, suppose that another datum arrives, and we are informed that Tweety might fly after all. Suppose further that our resource is not so sure about this information or that this resource is not a reliable one. We therefore have two options to express this uncertainty in our program: One option is to attach to the new information a low priority. Alternatively, we can put an attenuation factor in the clause body. The impact of the former option is that in case of conflicts we prefer the complementary information and ignore the new data altogether, while the effect of the latter option is that we always consider the new data, but give it a lower weight when we draw our conclusions. According to the the second option the modified program may be the following: [22]

$$\mathcal{P}_3 = \mathcal{P}_2 \cup \{\ \texttt{fly(Tweety)} \overset{2}{\leftarrow} \texttt{dt}\ \}$$

Table 2 describes the iterative construction of $\nu^{\mathcal{P}_3}$.

[22] Where `dt` is a propositional constant that corresponds to the truth value *dt* in *NINE* (intuitively understood as "true by default").

Table 2: iterative construction of $\nu^{\mathcal{P}_3}$

function	`bird`	$\neg$`bird`	`penguin`	$\neg$`penguin`	`fly`	$\neg$`fly`	`reptile`	$\neg$`reptile`
$\mathsf{threshold}_1$	t	$\bot$	t	$\bot$	dt	$\bot$	$\bot$	$\bot$
belief_1	t	$\bot$	t	$\bot$	dt	$\bot$	$\bot$	$\bot$
val_1	t	$\bot$	t	$\bot$	dt	$\bot$	$\bot$	$\bot$
ν_1	t	f	t	f	dt	df	$\bot$	$\bot$
$\mathsf{threshold}_2$	t	$\bot$	t	$\bot$	t	t	$\bot$	t
belief_2	t	$\bot$	t	$\bot$	dt	t	$\bot$	t
val_2	t	$\bot$	t	$\bot$	dt	t	$\bot$	t
ν_2	t	f	t	f	of	ot	f	t

Again, after two iterations we reach a fixpoint, in which $\nu^{\mathcal{P}_3}(\texttt{fly(Tweety)}) = of$. The interpretation of this result depends on the logical bilattice under consideration (i.e., the choice of the prime bifilter in *NINE*):

- In $\mathcal{NIN}\mathcal{E}_{ot}$ the fixpoint values of `fly(Tweety)` and of $\neg$`fly(Tweety)` are both designated. This means that the new data, although being somewhat unreliable, caused an inconsistent belief regarding the flying abilities of Tweety. Nevertheless, the fact that `fly(Tweety)` is assigned *of* rather than $\top$ reflects the fact that $\mathcal{P}_3$ contains more evidence in favour of $\neg$`fly(Tweety)` rather than in favour of `fly(Tweety)`.
- In $\mathcal{NIN}\mathcal{E}_t$ the fixpoint value of $\neg$`fly(Tweety)` is designated, while the fixpoint value of `fly(Tweety)` is not. This means that despite the new datum we actually still believe that Tweety cannot fly. However, because of the new information, we are less certain than before (thus `fly(Tweety)` is assigned *of* rather than f).

5 Some concluding remarks

We conclude with a summary of the main properties of the formalizms considered here, and some further remarks regarding related semantics.

Reasoning with incomplete and inconsistent data

One of the main drawbacks of some of the fixpoint semantics for extended logic program (like those in [26] and [36]) is that they are reduced to triviality in the presence of contradictions. As such, these formalisms inherit one of the well-known shortcomings of classical logic. We do believe that since inconsistent knowledge can and may be represented in extended logic programs, a plausible semantics for such programs should be able to handle inconsistent situations in a non-trivial way. That is, one should be able to draw meaningful conclusions (and reject others) despite the inconsistency. The fixpoint semantics considered here has such capabilities: It pinpoints on

the inconsistent and the incomplete parts of the data, and regards the rest of the information as classically consistent.

Consider, for instance, the following program, which is an extended variant of the program considered in Example 1.3 (see also item 3 of Example 3.18):

$$\mathcal{P} \;=\; \{\; p \leftarrow \mathtt{t} \qquad \neg p \leftarrow \mathtt{not}\; q \qquad r_1 \leftarrow \mathtt{t} \qquad r_2 \leftarrow \mathtt{not}\; \neg r_1 \qquad \neg r_3 \leftarrow r_1, \neg r_2 \;\}$$

This program provides a complete information regarding the truth or the falsity of r_i, $i = 1, 2, 3$. Moreover, the information regarding these atoms is not affected by either p, q, or their negations. The fact that the data regarding q is incomplete and the data regarding p is inconsistent should be *localized* (i.e., restricted only to those literals whose definitions depend on p or q), and it should *not* affect the values of the r_i's. Thus, the inconsistent data in $\mathcal{P}$ should not spoil the whole piece of information represented in this program. The fixpoint semantics that was considered here follows these guidelines; The unique adequate solution for $\mathcal{P}$ (and so its most adequate model) is the following:

$$\nu^{\mathcal{P}} \;=\; \{\; p\!:\!\top, \quad q\!:\!\bot, \quad r_1\!:\!t, \quad r_2\!:\!t, \quad r_3\!:\!f \;\}$$

It follows that the complete information in $\mathcal{P}$ (the one that concerns with r_i, $i = 1, 2, 3$) is preserved. In addition, the reasoner may realize that the data about p is contradictory, and the data about q is incomplete.

Relating negative data to its positive counterpart

Another major difference between the semantics introduced here and some other semantics for extended logic programs (e.g, [24, 26, 30, 41]) concerns with the way a negative data is related to its positive counterpart. While the formalizms of [24, 26, 30, 41] treat p and $\neg p$ as two different *atomic formulae*, we preserve the relation between an atomic formula and its negated atom. To see the importance of this, consider the following program (also considered in [9, Example 3.3.6] and [36, Example 1]):

$$\mathcal{P} \;=\; \{\; p \leftarrow \mathtt{not}\; q \qquad q \leftarrow \mathtt{not}\; p \qquad \neg p \leftarrow \mathtt{t} \;\}$$

According to the approaches that treat $\neg p$ as (a strange way of writing) an atomic formula, the well-founded semantics would assign here t to $\neg p$, $\bot$ to p, and $\bot$ to q. So even though $\mathcal{P}$ is classically consistent, the distinction between p and $\neg p$ causes a counter-intuitive result here. In contrast, our approach yields a semantics that seems to reflect the intuitive expectation in this case: The unique adequate solution for $\mathcal{P}$ assigns f to p and t to q.

For another example, consider the following logic program of [36, Example 6]:

$$\mathcal{P} \;=\; \{\; r \leftarrow \mathtt{not}\; q \qquad q \leftarrow \mathtt{not}\; p \qquad p \leftarrow \mathtt{not}\; p \qquad \neg q \leftarrow \mathtt{t} \;\}$$

The unique adequate solution for $\mathcal{P}$ (and so its most adequate model) is $\{p\!:\!\bot,\ q\!:\!f,\ r\!:\!t\}$ (which is the same as the one that is obtained in [36]). By considering $\neg q$ as a new atom, this program

would have a single extended stable model, in which $\neg q$ is true and all the other atomic formulae (p, q, r) are unknown. This seems to be a counter-intuitive result in this case, since one expects here that r would follow from $\mathcal{P}$.

Paraconsistent and coherent approaches to inconsistency

The formalisms that we have described here for giving semantics to extended logic programs are paraconsistent in nature. I.e., they accept contradictions within the theory and try to cope with them. Another common approach to handle contradictions (sometimes called *coherent* or *conservative* [12, 47]) first detects and eliminates the inconsistent part of the theory. Then, when consistency is restored, classical logic is used for drawing plausible conclusions from the "recovered" data. In [30], for instance, clauses with negative literals in their heads are getting a higher priority than clauses with positive literals in their head. The latter ones are ignored in case of contradictions with their negated counterparts. This approach assures a contradictions-free semantics [30, Theorem 2]. In [37] contradictions are excluded already in the level of knowledge representation, since clauses for default rules have the form $l \leftarrow Body, \texttt{not}\ \overline{l}$. Thus, in order to derive l, one has to verify first that its complement, $\overline{l}$, is not provable. Other coherent formalisms for managing inconsistent information are considered, e.g., in [2, 3, 4, 12, 16].

Flexible belief revision

Consider the following example (anonymous author):

> "A man fell from a plane. Fortunately, he was wearing a parachute. Unfortunately, the parachute didn't open. Fortunately, he fell from the plane at a low altitude over a large haystack. Unfortunately, there was a pitchfork in the haystack. Fortunately, he missed the pitchfork. Unfortunately, he missed the haystack ...".

After each sentence in this example there is a tendency to jump back and forth between opposite conclusions regarding the ultimate fate of the skydriver. In Tweety dilemma, considered in Section 4.4, we faced the same phenomenon when we had to change our mind several times regarding Tweety ability to fly in light of the new data that has arrived. Indeed, in the notations used in that section,

- $\texttt{fly(Tweety)}$ follows from $\mathcal{P}_2 \setminus \{\texttt{penguin(Tweety)} \overset{2}{\leftarrow} \texttt{t}\}$,
- $\texttt{fly(Tweety)}$ does not follow from $\mathcal{P}_2$,
- $\texttt{fly(Tweety)}$ does not follow from $\mathcal{P}_2 \cup \{\texttt{fly(Tweety)} \overset{2}{\leftarrow} \texttt{dt}\}$ w.r.t. $\mathcal{NIN}\mathcal{E}_t$ (i.e., in a skeptical reasoning), and it does follow from $\mathcal{P}_2 \cup \{\texttt{fly(Tweety)} \overset{2}{\leftarrow} \texttt{dt}\}$ w.r.t. $\mathcal{NIN}\mathcal{E}_{ot}$ (i.e., in a more liberal reasoning).

The need to alter the set of conclusions according to an input that is frequently modified is not an unusual phenomenon in commonsense reasoning in general and logic programming in particular. Thus, the plausibility of different formalisms in these areas is often determined by the way they

handle revised information. As demonstrated in Section 4.4, the flexibility of process for belief revision in our case is reflected both on the semantical level (different choices of logical bilattices yield different conclusions), and on the syntactical level (by enhancing the expressive power of the logic programs under consideration, thus allowing various ways to represent knowledge, either in the program language itself, or in a meta-language that reflects the reasoner's preferences).

Acknowledgement

I would like to thank Maurice Bruynooghe and Marc Denecker for their helpful comments on early versions of this paper. This work was supported by the visiting postdoctoral fellowship FWO Flanders.

Appendix A. Proofs

Proposition 3.3: Let $\mathcal{P}$ be a general logic program. Then $\nu^{\mathcal{P}}$ is the k-minimal four-valued model of $\mathcal{P}$. Moreover, it is at least as consistent as any other model of $\mathcal{P}$, and the consequence relation induced by it is paraconsistent.

Proof: This proposition contains several claims. We divide the proof accordingly.

1. *$\nu^{\mathcal{P}}$ is a model of $\mathcal{P}$*:
   Suppose not. Then there is a clause $l \leftarrow Body$ in $\mathcal{P}$ s.t. $\nu^{\mathcal{P}}(Body) \in \mathcal{D}$ while $\nu^{\mathcal{P}}(l) \notin \mathcal{D}$. In particular, there is an α s.t. for every $\beta \geq \alpha$, $\nu^{\mathcal{P}}_{\beta}(Body) \in \mathcal{D}$ while $\nu^{\mathcal{P}}_{\beta}(l) \notin \mathcal{D}$. But since $\nu^{\mathcal{P}}_{\alpha}(Body) \in \mathcal{D}$, for every $\beta > \alpha$, $\mathsf{val}^{\mathcal{P}}_{\beta}(l) = t$, which implies that $\nu^{\mathcal{P}}_{\beta}(l) \geq_k t$, and so $\nu^{\mathcal{P}}_{\beta}(l) \in \mathcal{D}$. This contradicts the assumption that $\nu^{\mathcal{P}}_{\beta}(l) \notin \mathcal{D}$.

2. *$\nu^{\mathcal{P}}$ induces a paraconsistent consequence relation*:
   Consider, e.g., $\mathcal{P} = \{p \leftarrow \mathtt{t},\ \neg p \leftarrow \mathtt{t}\}$. Here $\nu^{\mathcal{P}}(p) = \top$ while $\nu^{\mathcal{P}}(q) = \bot$ for every atom $q \neq p$. Thus $\mathcal{P} \not\sim_{\nu} q$ for every atom $q \neq p$, which means that trivial reasoning from an inconsistent set of premises is not allowed.

3. *$\nu^{\mathcal{P}}$ is $\leq_k$-smaller than any other model of $\mathcal{P}$*:
   For a valuation ν denote $\mathsf{Sat}(\nu) = \{l \mid \nu(l) \in \mathcal{D}\}$. Let α be the fixpoint ordinal of $\nu^{\mathcal{P}}$ (i.e., the minimal ϵ s.t. $\nu^{\mathcal{P}}_{\beta'} = \nu^{\mathcal{P}}_{\beta}$ for every $\beta' \geq \beta \geq \epsilon$). We show that for every model M of $\mathcal{P}$, $\mathsf{Sat}(\nu^{\mathcal{P}}_{\alpha}) \subseteq \mathsf{Sat}(M)$. This immediately implies that $\nu^{\mathcal{P}} \leq_k M$, since in this case, for every atom p, we have that

   - If $\nu^{\mathcal{P}}(p) = \top$, then $p, \neg p \in \mathsf{Sat}(\nu^{\mathcal{P}}_{\alpha})$. Thus $p, \neg p \in \mathsf{Sat}(M)$, and so $M(p) = \top$ as well.
   - If $\nu^{\mathcal{P}}(p) = t$, then $p \in \mathsf{Sat}(\nu^{\mathcal{P}}_{\alpha})$, and so $p \in \mathsf{Sat}(M)$. Thus $M(p) \in \{t, \top\}$, which implies that $M(p) \geq_k \nu^{\mathcal{P}}(p)$.
   - The case $\nu^{\mathcal{P}}(p) = f$ is similar to the one in which $\nu^{\mathcal{P}}(p) = t$.
   - If $\nu^{\mathcal{P}}(p) = \bot$ then clearly $M(p) \geq_k \nu^{\mathcal{P}}(p)$.

It remains to show, therefore, that for every model M of $\mathcal{P}$, $\mathsf{Sat}(\nu^{\mathcal{P}}_{\alpha}) \subseteq \mathsf{Sat}(M)$. We show this by an induction on α. The case $\alpha = 0$ is obvious, since $\mathsf{Sat}(\nu^{\mathcal{P}}_{0}) = \emptyset$. For $\alpha > 0$, let $l \in \mathsf{Sat}(\nu^{\mathcal{P}}_{\alpha})$. Then $\nu^{\mathcal{P}}_{\alpha}(l) \in \{t, \top\}$, and so $\mathsf{val}^{\mathcal{P}}_{\alpha}(l) \oplus \neg\mathsf{val}^{\mathcal{P}}_{\alpha}(\bar{l}) \in \{t, \top\}$. This means that either $\mathsf{val}^{\mathcal{P}}_{\alpha}(l) \in \{t, \top\}$, or $\neg\mathsf{val}^{\mathcal{P}}_{\alpha}(\bar{l}) \in \{t, \top\}$ (i.e., $\mathsf{val}^{\mathcal{P}}_{\alpha}(\bar{l}) \in \{f, \top\}$). But since for every literal l', $\mathsf{val}^{\mathcal{P}}_{\alpha}(l') \in \{t, \bot\}$, this means that in our case necessarily $\mathsf{val}^{\mathcal{P}}_{\alpha}(l) = t$. Hence, there is a general clause of the form $l \leftarrow Body$, for which $\nu^{\mathcal{P}}_{\alpha-1}(Body) \in \{t, \top\}$. Thus, for every $l_i \in Body$, $l_i \in \mathsf{Sat}(\nu^{\mathcal{P}}_{\alpha-1})$. By the induction hypothesis, for every $l_i \in Body$, $l_i \in \mathsf{Sat}(M)$, and so $M(Body) \in \{t, \top\}$ as well. But M is a model of $\mathcal{P}$, and so necessarily $M(l) \in \{t, \top\}$, i.e. $l \in \mathsf{Sat}(M)$.

4. *$\nu^{\mathcal{P}}$ is at least as consistent as any other model of $\mathcal{P}$*:
Denote again by α the fixpoint ordinal of $\nu^{\mathcal{P}}$, and let $\mathsf{Sat}(\nu) = \{l \mid \nu(l) \in \mathcal{D}\}$. Suppose that $\nu^{\mathcal{P}}(p) = \top$ for some atom p. Then $p, \neg p \in \mathsf{Sat}(\nu^{\mathcal{P}})$, and so $p, \neg p \in \mathsf{Sat}(\nu^{\mathcal{P}}_{\alpha})$. By the proof of the previous item, for every model M of $\mathcal{P}$, $\mathsf{Sat}(\nu^{\mathcal{P}}_{\alpha}) \subseteq \mathsf{Sat}(M)$. Thus $p, \neg p \in \mathsf{Sat}(M)$, and so $M(p) = \top$ as well. □

Proposition 3.6: Let $\mathcal{P}$ be a positive logic program, and let $\mathcal{P}$' be the positive program obtained from $\mathcal{P}$ by replacing every implication connective by a material implication. Denote by $\nu^{\mathcal{P}}_{f/\bot}$ the valuation that is obtained from $\nu^{\mathcal{P}}$ by changing the $\bot$-assignments to f-assignments (I.e., for every atom p, if $\nu^{\mathcal{P}}(p) = \bot$ then $\nu^{\mathcal{P}}_{f/\bot}(p) = f$). Then:

1. $\mathcal{P}$ and $\mathcal{P}'$ have the same classical models (and thus the same least Herbrand model), and
2. $\nu^{\mathcal{P}}_{f/\bot}$ is the (unique) 2-valued minimal Herbrand model of $\mathcal{P}$ and $\mathcal{P}$'.

Proof: The first claim simply follows from the fact that the implication connective of Definition 2.4 is the same as the material implication on $\{t, f\}$. Regarding the other part, note first that since only atomic formulae appear in the clauses heads, for every atomic p and for every i we have that $\mathsf{val}^{\mathcal{P}}_{i}(\neg p) = \bot$, and therefore $\nu^{\mathcal{P}}_{i}(p) = \mathsf{val}^{\mathcal{P}}_{i}(p) \in \{t, \bot\}$. It follows that $\nu^{\mathcal{P}}$ assigns only values in $\{t, \bot\}$ to the atomic formulae (and so, for every literal l, $\nu^{\mathcal{P}}(l) \in \{t, f, \bot\}$). Now, let $p \leftarrow Body$ be a clause in $\mathcal{P}$. Since $\mathcal{P}$ is positive, then $\nu^{\mathcal{P}}(Body) = t$ iff all the atoms in $Body$ are assigned t by $\nu^{\mathcal{P}}$, iff all the atoms in $Body$ are assigned t by $\nu^{\mathcal{P}}_{f/\bot}$, iff $\nu^{\mathcal{P}}_{f/\bot}(Body) = t$. Similarly, $\nu^{\mathcal{P}}(Body) \in \{f, \bot\}$ iff there is an atomic formula in $Body$ that is assigned either $\bot$ or f by $\nu^{\mathcal{P}}$, iff there is an atomic formula in $Body$ that is assigned f by $\nu^{\mathcal{P}}_{f/\bot}$, iff $\nu^{\mathcal{P}}_{f/\bot}(Body) = f$. It follows that for every clause $\mathcal{C}$ in $\mathcal{P}$, $\nu^{\mathcal{P}}(\mathcal{C}) \in \mathcal{D}$ iff $\nu^{\mathcal{P}}_{f/\bot}(\mathcal{C}) \in \mathcal{D}$. Thus $\nu^{\mathcal{P}}_{f/\bot}$ is a 2-valued model of $\mathcal{P}$. It remains to show that $\nu^{\mathcal{P}}_{f/\bot}$ is $\leq_t$-minimal among the classical models of $\mathcal{P}$. Indeed, this follows from the fact that $\nu^{\mathcal{P}}$ is $\leq_k$-smaller than any other model of $\mathcal{P}$ (Proposition 3.3), and so the set of atomic formulae that are assigned t by $\nu^{\mathcal{P}}$ does not contain any corresponding set of a classical model of $\mathcal{P}$.[23] Thus, the set of atomic formulae that are assigned t by $\nu^{\mathcal{P}}_{f/\bot}$ does not contain any corresponding set of a classical model of $\mathcal{P}$ either, and so $\nu^{\mathcal{P}}_{f/\bot}$ is a $\leq_t$-minimal model among the classical models of $\mathcal{P}$. Since $\mathcal{P}$ has the same classical models as those of $\mathcal{P}'$ (item 1 of this proposition), we conclude that

[23] Indeed, if M is a classical model of $\mathcal{P}$ and p is an atom s.t. $\nu^{\mathcal{P}}(p) = t$ while $M(p) \neq t$, then $M(p) = f$, and so $\nu^{\mathcal{P}} \not\leq_k M$.

$\nu^{\mathcal{P}}_{f/\bot}$ is also a $\leq_t$-minimal model among the classical models of $\mathcal{P}'$. But being positive, $\mathcal{P}$' has only *one* $\leq_t$-minimal classical model (which is its least Herbrand model), and so $\nu^{\mathcal{P}}_{f/\bot}$ coincides with this model. □

Proposition 3.9: Let $\mathcal{P}$ be a normal logic program in which each atomic formula appears at most once in a clause head, and let $\Psi^{\mathcal{P}}$ be Fitting fixpoint semantics for $\mathcal{P}$. Then $\Psi^{\mathcal{P}} = \nu^{\mathcal{P}^*}$.

Proof: By Definition 3.8, $\mathcal{P}^* = \mathcal{P} \cup \mathcal{P}_\neg$ where

$$\mathcal{P}_\neg = \{\neg p \leftarrow \bar{l} \mid p \leftarrow Body \in \mathcal{P},\ \nu^{\mathcal{P}}(p) = \bot,\ l \in \mathcal{L}(Body)\} \cup$$
$$\{\neg p \leftarrow \mathtt{t} \mid p \leftarrow \mathtt{f} \in \mathcal{P},\ \nu^{\mathcal{P}}(p) = \bot\}.$$

• Suppose first that for some q, $\Psi^{\mathcal{P}}(q) = t$. We show that in this case $\nu^{\mathcal{P}}(q) = t$ as well. Assuming this, then by the definition of $\mathcal{P}_\neg$, $\neg q$ cannot appear in the head of any clause of $\mathcal{P}_\neg$, and so $\neg q$ cannot appear in the head of any clause of $\mathcal{P}^*$. It follows, then, that for every α, $\mathsf{val}^{\mathcal{P}^*}_\alpha(\neg q) = \bot$, and so $\nu^{\mathcal{P}^*}_\alpha(q) = \mathsf{val}^{\mathcal{P}^*}_\alpha(q) \in \{t, \bot\}$. Thus $\nu^{\mathcal{P}^*}(q) \in \{t, \bot\}$. On the other hand, if $\mathcal{P}_1 \subseteq \mathcal{P}_2$ then $\nu^{\mathcal{P}_2} \geq_k \nu^{\mathcal{P}_1}$, thus $\nu^{\mathcal{P}^*}(q) \geq_k \nu^{\mathcal{P}}(q) = t$. It follows, then, that $\nu^{\mathcal{P}^*}(q) = t$, and so $\nu^{\mathcal{P}^*}(q) = \Psi^{\mathcal{P}}(q)$ in this case.

To complete the proof for the first case it remains therefore to show that for every atom q, if $\Psi^{\mathcal{P}}(q) = t$ then $\nu^{\mathcal{P}}(q) = t$ as well. Let $\{\Psi^{\mathcal{P}}_0, \Psi^{\mathcal{P}}_1, \ldots\}$ be the $\leq_k$-monotonic iterative sequence of valuations used for constructing $\Psi^{\mathcal{P}}$. Since $\neg q$ does not appear in any clause head in $\mathcal{P}$, we have that for every α $\mathsf{val}^{\mathcal{P}}_\alpha(\neg q) = \bot$, and so $\nu^{\mathcal{P}}_\alpha(p) = \mathsf{val}^{\mathcal{P}}_\alpha(q) = t$. Thus, for showing that if $\Psi^{\mathcal{P}}(q) = t$ then $\nu^{\mathcal{P}}(q) = t$, it is sufficient to show that for every α and atom q s.t. $\Psi^{\mathcal{P}}_\alpha(q) = t$, $\mathsf{val}^{\mathcal{P}}_\alpha(q) = t$ as well. We show it by induction on α. For $\alpha = 0$ we have that $\Psi^{\mathcal{P}}_0(q) = \mathsf{val}^{\mathcal{P}}_\alpha(q) = \bot$, so the condition is vacuously met. For $\alpha = 1$, $\Psi^{\mathcal{P}}_1(q) = t$ iff $q \leftarrow \mathtt{t} \in \mathcal{P}$ iff $\mathsf{val}^{\mathcal{P}}_1(q) = t$. For $\alpha > 1$, $\Psi^{\mathcal{P}}_\alpha(q) = t$ iff there is a clause of the form $q \leftarrow Body$ in $\mathcal{P}$ and $\Psi^{\mathcal{P}}_{\alpha-1}(Body) = t$, iff $\forall l_i \in \mathcal{L}(Body)$ $\Psi^{\mathcal{P}}_{\alpha-1}(l_i) = t$, iff (induction hypothesis) $\forall l_i \in \mathcal{L}(Body)$ $\mathsf{val}^{\mathcal{P}}_{\alpha-1}(l_i) = t$. Thus $\mathsf{val}^{\mathcal{P}}_{\alpha-1}(Body) = t$, which implies that $\nu^{\mathcal{P}}_{\alpha-1}(Body) \in \mathcal{D}$, and so $\mathsf{val}^{\mathcal{P}}_\alpha(q) = t$.

• Suppose now that $\Psi^{\mathcal{P}}(q) = f$. Let $\{\Psi^{\mathcal{P}}_0, \Psi^{\mathcal{P}}_1, \ldots\}$ be the $\leq_k$-monotonic iterative sequence of valuations used for constructing $\Psi^{\mathcal{P}}$. We show that for every α and literal l s.t. $\Psi^{\mathcal{P}}_\alpha(l) = f$, $\mathsf{val}^{\mathcal{P}^*}_\alpha(\bar{l}) = t$. Assuming this, we are able to show that $\nu^{\mathcal{P}^*}(q) = \Psi^{\mathcal{P}}(q)$ in this case as well, since the fact that $\Psi^{\mathcal{P}}(q) = f$ implies that there exists some ϵ s.t. for every $\beta \geq \epsilon$ $\Psi^{\mathcal{P}}_\beta(q) = f$, and so by our assumption, $\mathsf{val}^{\mathcal{P}^*}_\beta(\neg q) = t$. Note also that by Proposition 3.3, $\nu^{\mathcal{P}}$ is the $\leq_k$-least model of $\mathcal{P}$, thus, since $\Psi^{\mathcal{P}}$ is a model of $\mathcal{P}$, and since $f = \Psi^{\mathcal{P}}(q) \geq_k \nu^{\mathcal{P}}(q) \in \{t, \bot\}$, necessarily $\nu^{\mathcal{P}}(q) = \bot$. Thus, for every α, $\mathsf{val}^{\mathcal{P}}_\alpha(q) = \bot$. Since q does not appear in the head of clauses in $\mathcal{P}_\neg$, this means that for every α, $\mathsf{val}^{\mathcal{P}^*}_\alpha(q) = \bot$ as well. It follows, then, that for every $\beta \geq \epsilon$ $\nu^{\mathcal{P}^*}_\beta(q) = \mathsf{val}^{\mathcal{P}^*}_\beta(q) \oplus \neg\mathsf{val}^{\mathcal{P}^*}_\beta(\neg q) = \bot \oplus \neg t = f$. One thus concludes that $\nu^{\mathcal{P}^*}(q) = f = \Psi^{\mathcal{P}}(q)$.

For the second case of the proof it remains, therefore, to show that for every α and a literal l s.t. $\Psi^{\mathcal{P}}_\alpha(l) = f$, $\mathsf{val}^{\mathcal{P}^*}_\alpha(\bar{l}) = t$. We show it by induction on α. For $\alpha = 0$ the condition is vacuously met, since $\Psi^{\mathcal{P}}_0(l) = \bot$ for every l. For $\alpha = 1$, the fact that $\Psi^{\mathcal{P}}_1(l) = f$ entails that $l \leftarrow \mathtt{f}$ appears in $\mathcal{P}$. Since $\mathcal{P}$ is a normal program, l must be an atom in this case. Moreover, by our assumption on $\mathcal{P}$, this is the only clause which contains l as its head, and so $\nu^{\mathcal{P}}(l) = \bot$. Thus $\bar{l} \leftarrow \mathtt{t}$ appears

in $\mathcal{P}_\neg$, and so $\mathsf{val}_1^{\mathcal{P}^*}(\overline{l}) = t$. Suppose now that for some $\alpha > 1$, $\Psi_\alpha^{\mathcal{P}}(l) = f$. By the construction of the $\Psi_j^{\mathcal{P}}$-s, and by our assumption on $\mathcal{P}$, it follows that for the only clause of the form $l \leftarrow Body$ that appears in $\mathcal{P}$, $\Psi_{\alpha-1}^{\mathcal{P}}(Body) = f$. This means that there is some $l' \in \mathcal{L}(Body)$ s.t. $\Psi_{\alpha-1}^{\mathcal{P}}(l') = f$. By induction hypothesis, then, $\mathsf{val}_{\alpha-1}^{\mathcal{P}^*}(\overline{l'}) = t$. Now, $\Psi^{\mathcal{P}}(l) \geq_k \Psi_\alpha^{\mathcal{P}}(l) = f$ thus $\Psi^{\mathcal{P}}(l) = f$. On the other-hand, using again the fact that $\nu^{\mathcal{P}}$ is the $\leq_k$-least model of $\mathcal{P}$ and that $\Psi^{\mathcal{P}}$ is a model of $\mathcal{P}$, $\Psi^{\mathcal{P}}(l) \geq_k \nu^{\mathcal{P}}(l) \in \{t, \bot\}$. Hence $\nu^{\mathcal{P}}(l) = \bot$. This means that $\overline{l} \leftarrow \overline{l'}$ appears in $\mathcal{P}_\neg$. But $\mathsf{val}_{\alpha-1}^{\mathcal{P}^*}(\overline{l'}) = t$, and so $\nu_{\alpha-1}^{\mathcal{P}^*}(\overline{l'})$ is designated. Thus, $\mathsf{val}_\alpha^{\mathcal{P}^*}(\overline{l}) = t$, as required.

• Finally, suppose that $\Psi^{\mathcal{P}}(q) = \bot$. Again, let $\{\Psi_0^{\mathcal{P}}, \Psi_1^{\mathcal{P}}, \ldots\}$ be the $\leq_k$-monotonic iterative sequence of valuations used for constructing $\Psi^{\mathcal{P}}$. This time we show that for every α and literal l s.t. $\Psi_\alpha^{\mathcal{P}}(l) = \bot$ we have that $\mathsf{val}_\alpha^{\mathcal{P}^*}(\overline{l}) = \bot$ as well. This implies that $\nu^{\mathcal{P}^*}(q) = \Psi^{\mathcal{P}}(q)$ also in this case, since the fact that $\Psi^{\mathcal{P}}(q) = \bot$ implies that for every α, $\Psi_\alpha^{\mathcal{P}}(q) = \bot$, and so by our assumption, for every α we have that $\mathsf{val}_\alpha^{\mathcal{P}^*}(\neg q) = \bot$. Note also that since $\bot = \Psi^{\mathcal{P}}(q) \geq_k \nu^{\mathcal{P}}(q)$, necessarily $\nu^{\mathcal{P}}(q) = \bot$, and so $\mathsf{val}_\alpha^{\mathcal{P}}(q) = \bot$ for every α. Since q does not appear in the head of clauses in $\mathcal{P}_\neg$, this means that for every α, $\mathsf{val}_\alpha^{\mathcal{P}^*}(q) = \bot$ as well. Thus $\nu^{\mathcal{P}^*}(q) = \mathsf{val}_\alpha^{\mathcal{P}^*}(q) \oplus \neg \mathsf{val}_\alpha^{\mathcal{P}^*}(\neg q) = \bot \oplus \neg \bot = \bot$. It follows that $\nu^{\mathcal{P}^*}(q) = \bot = \Psi^{\mathcal{P}}(q)$.

It remains to show that for every α and a literal l s.t. $\Psi_\alpha^{\mathcal{P}}(l) = \bot$, we have that also $\mathsf{val}_\alpha^{\mathcal{P}^*}(\overline{l}) = \bot$. We show it by induction on α. For $\alpha = 0$ this is obviously true, since by their definitions $\Psi_0^{\mathcal{P}}$ and $\mathsf{val}_0^{\mathcal{P}^*}$ are both identically $\bot$. For $\alpha = 1$, $\Psi_1^{\mathcal{P}}(l) = \bot$ iff $\Psi_1^{\mathcal{P}}(p) = \bot$ where p is the atomic part of l, iff either p does not appear in the head of any clause of $\mathcal{P}$, or $p \leftarrow Body \in \mathcal{P}$ and $\mathcal{L}(Body) \neq \emptyset$. In the first case neither l nor $\overline{l}$ appear in the head of any clause of $\mathcal{P}^*$, and in the second case if a clause of the form $\overline{l} \leftarrow Body$ appears in $\mathcal{P}^*$, then $\mathcal{L}(Body) \neq \emptyset$. In both cases, therefore, $\mathsf{val}_1^{\mathcal{P}^*}(\overline{l}) = \bot$. For $\alpha > 1$, $\Psi_\alpha^{\mathcal{P}}(l) = \bot$ means again that $\Psi_\alpha^{\mathcal{P}}(p) = \bot$, where p is the atomic part of l. This can happen if either p does not appear in the head of any clause of $\mathcal{P}$ (which again implies that $\mathsf{val}_\alpha^{\mathcal{P}^*}(\overline{l}) = \bot$, as in the basis of the induction), or else — by our assumption on $\mathcal{P}$ — there is a single clause in $\mathcal{P}$ of the form $p \leftarrow Body$ and $\Psi_{\alpha-1}^{\mathcal{P}}(Body) = \bot$. This means that $\forall l' \in \mathcal{L}(Body)$ $\Psi_{\alpha-1}^{\mathcal{P}}(l') \in \{t, \bot\}$ (and $\exists l'_0 \in \mathcal{L}(Body)$ s.t. $\Psi_{\alpha-1}^{\mathcal{P}}(l'_0) = \bot$). By what we have show in the first case of this proof (in case that $\Psi_{\alpha-1}^{\mathcal{P}}(l') = t$) and by the induction hypothesis (in case that $\Psi_{\alpha-1}^{\mathcal{P}}(l') = \bot$), $\mathsf{val}_{\alpha-1}^{\mathcal{P}^*}(\overline{l'}) = \bot$ for every $l' \in \mathcal{L}(Body)$. Thus, $\nu_{\alpha-1}^{\mathcal{P}^*}(l') = \mathsf{val}_{\alpha-1}^{\mathcal{P}^*}(l') \in \{\bot, t\}$. In other words, $\nu_{\alpha-1}^{\mathcal{P}^*}(\overline{l'}) \in \{\bot, f\}$ so $\nu_{\alpha-1}^{\mathcal{P}^*}(\overline{l'})$ is not designated. Since the only clauses in which $\neg p$ may appear as their head are of the form $\neg p \leftarrow \overline{l'}$, it follows that $\mathsf{val}_\alpha^{\mathcal{P}^*}(\neg p) = \bot$. Since p do not appear as a head of any clause in $\mathcal{P}_\neg$, we also have that $\mathsf{val}_\alpha^{\mathcal{P}^*}(p) = \mathsf{val}_\alpha^{\mathcal{P}}(p) \leq_k \nu_\alpha^{\mathcal{P}}(p) \leq_k \nu^{\mathcal{P}}(p) \leq_k \Psi^{\mathcal{P}}(p) = \bot$. Hence, both $\mathsf{val}_\alpha^{\mathcal{P}^*}(p) = \bot$ and $\mathsf{val}_\alpha^{\mathcal{P}^*}(\neg p) = \bot$. Thus, either if $l = p$ or $l = \neg p$, we have that $\mathsf{val}_\alpha^{\mathcal{P}^*}(\overline{l}) = \bot$. □

Proposition 4.14: Let $\mathcal{P}$ be a general logic program, and let $\mathcal{P}'$ be the 1-prioritized logic program obtained from $\mathcal{P}$ by assigning the quantitative factor $n = 1$ to every general clause in $\mathcal{P}$. Then, for every i, $\nu_i^{\mathcal{P}}$ (defined in 3.1) is the same as $\nu_i^{\mathcal{P}'}$ (defined in 4.12).

Proof: First, as noted in Example 4.7, the bilattice $\mathcal{B} = \{0,1\} \odot \{0,1\}$ that gives semantics to the 1-prioritized program $\mathcal{P}'$ is isomorphic to the bilattice $\mathcal{FOUR}$ used in Section 2 to give semantics to the "flat" (non-prioritized) program $\mathcal{P}$. In what follows we shall use both representations to denote the same elements.

By the definition of the $\leq_t$-operations and the $\leq_k$-operations in the bilattice $\{0,1\}\odot\{0,1\}$, we have that [24]

$$\begin{aligned}\nu_i^{\mathcal{P}}(l) &= \mathsf{val}_i^{\mathcal{P}}(l) \oplus \neg\mathsf{val}_i^{\mathcal{P}}(\bar{l}) = ([\mathsf{val}_i^{\mathcal{P}}(l)]_T, [\mathsf{val}_i^{\mathcal{P}}(l)]_F) \oplus ([\mathsf{val}_i^{\mathcal{P}}(\bar{l})]_F, [\mathsf{val}_i^{\mathcal{P}}(\bar{l})]_T) \\ &= (max([\mathsf{val}_i^{\mathcal{P}}(l)]_T, [\mathsf{val}_i^{\mathcal{P}}(\bar{l})]_F),\ max([\mathsf{val}_i^{\mathcal{P}}(l)]_F, [\mathsf{val}_i^{\mathcal{P}}(\bar{l})]_T)).\end{aligned}$$

But since $\mathsf{val}_i^{\mathcal{P}}(\cdot) \in \{t, \bot\} = \{(1,0),(0,0)\}$, we have that $[\mathsf{val}_i^{\mathcal{P}}(\cdot)]_F = 0$, thus

$$\nu_i^{\mathcal{P}}(l) = ([\mathsf{val}_i^{\mathcal{P}}(l)]_T, [\mathsf{val}_i^{\mathcal{P}}(\bar{l})]_T).$$

Since $\nu_i^{\mathcal{P}'}(l) = ([\mathsf{val}_i^{\mathcal{P}'}(l)]_T, [\mathsf{val}_i^{\mathcal{P}'}(\bar{l})]_T)$, it remains to show that for every i and l, $[\mathsf{val}_i^{\mathcal{P}}(l)]_T = [\mathsf{val}_i^{\mathcal{P}'}(l)]_T$. Indeed, recall that all the clauses in $\mathcal{P}'$ are assigned the maximal confidence factor (which is 1 in our case), and so for every i and l, $[\mathsf{threshold}_i^{\mathcal{P}}(\bar{l})]_T \leq 1$. It follows, then, that

$$\begin{aligned}\mathsf{val}_i^{\mathcal{P}'}(l) &= lub_{\leq_k}(\mathsf{val}_{i-1}^{\mathcal{P}'}(l),\ \mathsf{belief}_i^{\mathcal{P}'}(l)) \\ &= lub_{\leq_k}(\mathsf{val}_{i-1}^{\mathcal{P}'}(l),\ lub_{\leq_k}\{\nu_{i-1}^{\mathcal{P}'}(Body_j) \mid l \overset{1}{\leftarrow} Body_j \in \mathcal{P}'\}).\end{aligned}$$

Using again the fact that $\nu_i^{\mathcal{P}'}(l) = ([\mathsf{val}_i^{\mathcal{P}'}(l)]_T, [\mathsf{val}_i^{\mathcal{P}'}(\bar{l})]_T)$, we have that

$$\begin{aligned}[\mathsf{val}_i^{\mathcal{P}'}(l)]_T &= \max([\mathsf{val}_{i-1}^{\mathcal{P}'}(l)]_T,\ \max\{[\nu_{i-1}^{\mathcal{P}'}(Body_j)]_T \mid l \overset{1}{\leftarrow} Body_j \in \mathcal{P}'\}) \\ &= \max([\nu_{i-1}^{\mathcal{P}'}(l)]_T,\ \max\{[\nu_{i-1}^{\mathcal{P}'}(Body_j)]_T \mid l \overset{1}{\leftarrow} Body_j \in \mathcal{P}'\}).\end{aligned}$$

Since in our case $\mathcal{B} = \{0,1\}\odot\{0,1\}$, it follows that

$[\mathsf{val}_i^{\mathcal{P}'}(l)]_T = 1$ if $\exists(l \overset{1}{\leftarrow} Body) \in \mathcal{P}'$ and $\nu_{i-1}^{\mathcal{P}'}(Body) \in \mathcal{D}_{\{0,1\}\odot\{0,1\}}$,

$[\mathsf{val}_i^{\mathcal{P}'}(l)]_T = 0$ otherwise.

On the other hand, by the definition of $\mathsf{val}_i^{\mathcal{P}}$,

$[\mathsf{val}_i^{\mathcal{P}}(l)]_T = 1$ if $\mathsf{val}_i^{\mathcal{P}}(l) = t$, if $\exists(l \leftarrow Body) \in \mathcal{P}$ and $\nu_{i-1}^{\mathcal{P}}(Body) \in \mathcal{D}_{\mathcal{FOUR}}$,

$[\mathsf{val}_i^{\mathcal{P}}(l)]_T = 0$ otherwise.

It follows, therefore, that $[\mathsf{val}_i^{\mathcal{P}}(l)]_T = [\mathsf{val}_i^{\mathcal{P}'}(l)]_T$, as required. □

References

[1] J.J.Alferes, H.Herre, L.M.Pereira. *Partial models of extended generalized logic programs.* Proc. 1st International Conference on Computational Logic (CL'2000), Lecture Notes in Artificial Intelligence 1861, (J.Lloyd et al., editors), pages 149–163, Springer-Verlag, 2000.

[2] J.J.Alferes, J.A.Leite, L.M.Pereira, H.Przymusinska, T.C.Przymusinski. *Dynamic updates of non-monotonic knowledge bases.* Journal of Logic Programming 45, pages 43–70, 2000.

[24] See the proof of Proposition 4.8 for the definitions of the relevant operations.

[3] O.Arieli. *Four-valued logics for reasoning with uncertainty in prioritized data.* Information, Uncertainty, Fusion (B.Bouchon-Meunier, R.R.Yager, L.A.Zadeh, editors) pages 293–304, Kluwer Academic Publishers, 1999.

[4] O.Arieli. *An algorithmic approach to recover inconsistent knowledge-bases.* To appear in: Proc. 7th European Workshop on Logics in Artificial Intelligence (Jelia'00), Lecture Notes in Artificial Intelligence 1919 (M.Ojeda-Aciego, I.P.de Guzman, G.Brewka, L.M.Pereira, editors), Springer-Verlag, 2000.

[5] O.Arieli, A.Avron. *Reasoning with logical bilattices.* Journal of Logic, Language, and Information 5(1), pages 25–63, 1996.

[6] O.Arieli, A.Avron. *The value of the four values.* Artificial Intelligence 102(1), pages 97–141, 1998.

[7] A.Avron. *Simple consequence relations.* Journal of Information and Computation 92, pages 105–139, 1991.

[8] A.Avron. *The structure of interlaced bilattices.* Journal of Mathematical Structures in Computer Science 6, pages 287–299, 1996.

[9] C.Baral, M.Gelfond. *Logic programming and knowledge representation.* Journal of Logic Programming 19–20, pages 73–148, 1994.

[10] N.D.Belnap. *A useful four-valued logic.* Modern Uses of Multiple-Valued Logic (G.Epstein, J.M.Dunn, editors), pages 7–37, Reidel Publishing Company, 1977.

[11] N.D.Belnap. *How computer should think.* Contemporary Aspects of Philosophy (G.Ryle, editor), pages 30–56, Oriel Press, 1977.

[12] S.Benferhat, D.Dubois, H.Prade. *How to infer from inconsistent beliefs without revising?* Proc. International Joint Conference on Artificial Intelligence (IJCAI'95), pages 1449–1455, 1995.

[13] K.L.Clark. *Negation as failure.* Logic and Databases (H.Gallaire, J.Minker, editors), pages 293–322, Plenum Press, 1978.

[14] N.C.A.da-Costa. *On the theory of inconsistent formal systems.* Notre Damm Journal of Formal Logic 15, pages 497–510, 1974.

[15] M.Denecker. *The well-founded semantics is the principle of inductive definitions.* Proc. 6th European Workshop on Logic in Artificial Intelligence (Jelia'98), (J.Dix, L.Fari nas del Cerro, U.Furbach, editors), Lecture Notes in Artificial Intelligence 1498, pages 1–16, Springer-Verlag, 1998.

[16] D.Dubios, J.Lang, H.Prade. *Possibilistic logic.* Handbook of Logic in Artificial Intelligence and Logic Programming (D.Gabbay, C.Hogger, J.Robinson, edtors), pages 439–513, Oxford Science Publications, 1994.

[17] C.V.Damasio, L.M.Pereira. *A model theory for paraconsistent logic programming.* Proc. 7th Portuguese Conference on Artificial Intelligence, Lecture Notes in Artificial Intelligence 990, pages 409–418, Springer-Verlag, 1995.

[18] M.Fitting. *Kripke-Kleene semantics for logic programs.* Journal of Logic Programming 2, pages 295–312, 1985.

[19] M.Fitting. *Bilattices in logic programming.* Proc. 20th IEEE International Symposium on Multiple-valued Logics (G.Epstein, editor), pages 238–246, IEEE Press, 1990.

[20] M.Fitting. *Kleene's logic, generalized.* Journal of Logic and Computation 1, pages 797–810, 1990.

[21] M.Fitting. *Bilattices and the semantics of logic programming.* Journal of Logic Programming 11(2), pages 91–116, 1991.

[22] M.Fitting. *The family of stable models.* Journal of Logic Programming 17, pages 197–225, 1993.

[23] M.Fitting. *Kleene's three-valued logics and their children.* Fundamenta Informaticae 20, pages 113–131, 1994.

[24] A.J.García, G.R.Simari, C.I.Chesñevar. *An argumentative framework for reasoning with inconsistent and incomplete information.* ECAI'98 Workshop on Practical Reasoning and Rationality, 1998.

[25] M.Gelfond, V.Lifschitz. *The stable model semantics for logic programming.* Proc. 5th Logic Programming Symposium (R.Kowalski, K.Bowen, editors), pages 1070–1080, MIT Press, 1988.

[26] M.Gelfond, V.Lifschitz. *Logic programs with classical negation.* Proc. 7th International Conference on Logic Programming (ICLP'90), (D.Warren, P.Szeredi, editors), pages 579–597, 1990.

[27] M.L.Ginsberg. *Multi-valued logics.* Readings in Non-Monotonic Reasoning (M.L.Ginsberg, editor), pages 251–258, 1987.

[28] M.L.Ginsberg. *Multi-valued logics: A uniform approach to reasoning in artificial intelligence.* Computer Intelligence 4, pages 256–316, 1988.

[29] S.C.Kleene. *Introduction to metamathematics.* Van Nostrand, 1950.

[30] R.A.Kowalski, F.Sadri. *Logic programs with exceptions.* Proc. 7th International Conference on Logic Programming (ICLP'90), (D.Warren, P.Szeredi, editors), pages 598–613, 1990.

[31] S.Kraus, D.Lehmann, M.Magidor. *Nonmonotonic reasoning, preferential models and cumulative logics.* Artificial Intelligence 44, pages 167–207, 1990.

[32] J.W.Lloyd. *Foundations of logic programming.* Springer Verlag, 1987.

[33] D.Makinson. *General patterns in nonmonotonic reasoning.* Handbook of Logic in Artificial Intelligence and Logic Programming 3 (D.Gabbay, C.Hogger, J.Robinson, editors), pages 35–110, Oxford Science Publications, 1994.

[34] B.Messing. *Combining knowledge with many-valued logics.* Journal of Data and Knowledge Engineering 23, pages 297–315, 1997.

[35] R.Nelken, N.Francez. *Bilattices and the semantics of natural language questions.* Technical Report, Department of Computer Science, The Technion, Israel, 1998.

[36] L.M.Pereira, J.J.Alferes. *Well-founded semantics for logic programs with explicit negation.* Proc. 10th European Conference on Artificial Intelligence (ECAI'92), (B.Neumann, editor), pages 102–106, 1992.

[37] L.M.Pereira, J.N.Aparício, J.J.Alferes. *Nonmonotonic reasoning with well founded semantics.* Proc. 8th International Conference on Logic Programming (ICLP'91), (K.Furukawa, editor), pages 475–489, 1991.

[38] G.Priest. *Minimally Inconsistent LP.* Studia Logica 50, pages 321–331, 1991.

[39] H.Przymusinska, T.Przymusinski. *Weakly perfect model semantics for logic programs.* Proc. 5th Logic Programming Symposium (R.Kowalski, K.Bowen, editors), pages 1106–1122, 1988.

[40] H.Przymusinska, T.Przymusinski. *Semantic issues in deductive databases and logic programs.* Formal Techniques in Artificial Intelligence (R.B.Banejí, editor), pages 321–367, Elsevier Science Publishers, 1990.

[41] T.Przymusinski. *Extended stable semantics for normal and disjunctive programs.* Proc. 7th International Conference on Logic Programming (ICLP'90), (D.Warren, P.Szeredi, editors), pages 459–477, 1990.

[42] R.Reiter *On closed word data bases.* Logic and Data Bases (H.Gallaire, J.Minker, editors), pages 119–140, Plenum Press, 1978.

[43] A.Schoter. *Evidential bilattice logic and lexical inferences.* Journal of Logic, Language and Information 5(1), pages 65–105, 1996.

[44] A.Tarski. *Lattice-theoretic fixpoint theorem and its applications.* Pacific Journal of Mathematics 5, pages 285–309, 1955.

[45] M.van-Emden, R.A.Kowalski. *The semantics of predicate logic as a programming language.* Journal of the ACM 23(4), pages 733–742, 1976.

[46] A.Van Gelder, K.A.Ross, J.S.Schlipf. *The well-founded semantics for general logic programs.* Journal of the ACM 38(3), pages 620–650, 1991.

[47] G.Wagner. *Vivid logic.* Lecture Notes in Artificial intelligence 764, Springer-Verlag, 1994.